Multi-ASIP architectures for flexible turbo receiver

Atif Raza Jafri

N° d'ordre : 2011telb0170

Sous le sceau de l'Université européenne de Bretagne

Télécom Bretagne

En habilitation conjointe avec l'Université de Bretagne-Sud

Ecole Doctorale – sicma

Architectures multi-ASIP pour turbo récepteur flexible

Thèse de Doctorat

Mention : STIC (Sciences et Technologies de l'Information et de la Communication)

Présentée par **Atif Raza Jafri**

Département : Electronique

Laboratoire : Lab-STICC / Pôle CACS

Directeur de thèse : Michel Jézéquel

Soutenue le 28 janvier 2011

Jury :

M. Guido Massera,	Professeur à Politecnico di Torino	(Rapporteur)
M. Olivier Sentieys,	Professeur à l'ENSSAT	(Rapporteur)
M. Guy Gogniat,	Professeur à l'Université de Bretagne-Sud	(Président)
M. Smail Niar,	Professeur à l'Université de Valenciennes	(Examinateur)
M. Christophe Moy,	Professeur à SUPELEC	(Examinateur)
M. Pierre Pénard,	Ingénieur recherche et développement à Orange Labs	(Examinateur)
M. Michel Jézéquel,	Professeur à Télécom Bretagne	(Directeur)
M. Amer Baghdadi,	Maître de conférences à Télécom Bretagne	(Encadrent)

Dedication

To my family.

Contents

List of Figures

List of Tables

List of Sample Programs

Introduction

LAST three decades can undoubtedly be said as the decades for wireless communications. New data transmission techniques have been evolved in this period of time to meet a goal which can be phrased as "achieve high data transmission rate at excessively low error rate using low transmission power and minimal channel bandwidth". In pursuit of achieving this goal, new state of the art data processing techniques have been developed in this field. The new state of the art techniques target, between others, the Error Control Coding (ECC) or channel coding, Bit Interleaved Coded Modulation (BICM), high ordered Quadrature Amplitude Modulations (QAM), Signal Space Diversity (SSD), Multi Input Multi Output (MIMO).

The data processing before the transmission of source data in the channel includes addition of redundant information in the original data (addition of parity during ECC), and/or rearrangement of data stream (carried out through BICM) and/or addition of diversity of data (incorporated through Space Time Code of MIMO). At the inputs of the receiver, the received information is corrupted by the destructive effects of the channel. Inside the receiver, multiple processing blocks work together to extract the original source data from the received corrupted data. These processing blocks use the added redundancy and/or diversity to recover the original data. Before the invention of Turbo Codes, each processing block of the receiver was used to process the data once and the output was passed to the next processing block. Invention of Turbo Codes introduced the iterative processing concept in the channel decoding block of the baseband receiver. With iterative processing the output results are improved over the iterations and hence using turbo codes error rate performance close to theoretical limits can be achieved. Based on the same iterative concept, other turbo principles are developed by feeding the information back to other processing block like the demodulator and the equalizer. These concepts, also known as turbo demodulation and turbo equalization, provide benefits when the channel adds fading and inter symbol interference effects. Although turbo processing provides improvements in error rate performance, it induces high processing latency and reduces the throughput of the receiver.

In the commercial world, using these evolved data transmission techniques in wireless communication domain, services such as cellular telephone networks, local and wide area networks, digital video broadcast are introduced. In order to guide the industry on using these state of the art techniques for different wireless communication applications, different wireless communication standards have been emerged in recent times such as: UMTS, 3GPP2, 3GPP-LTE for mobile phones, 802.11 (WiFi) and 802.16 (WiMax) for wireless local and wide area networks, DVB-RCS, DVB-S2, DVB-T2 for digital video broadcasting. Moreover, these wireless standards propose different parameters of above-stated state of the art techniques to provide best performance under particular transmission environment. Parameterization of individual components of the transmitter imposes the requirement of flexibility of architecture both in the transmitter and receiver. Hence, this emerging flexibility need in digital baseband design constitutes a new major challenge when added to the ever increasing requirements in terms of high throughput and low area for an iterative receiver.

Problems and Objectives

The presented context raises the question of how to implement flexible, yet high throughput digital communication systems which can take the benefits of turbo/iterative processing? In addition, how an architecture can provide the demanded processing power depending upon the parameters for which the system has to be configured? Heterogeneous MPSoCs (Multi Processor System on Chip) is one of the promising architectural solution to answer these requirements. This architecture model provides multiple advantages: modularity of the system to control its complexity, programmability for the adaptation of the system to the context of utilization and finally scalability of the system according to processing power demands. The next step is to show how to integrate and adapt appropriately the underlined processing, memory and communication units to modern wireless communication applications especially turbo communications.

Contributions

Towards this objective, following are the proposed contribution of this thesis work in algorithmic domain and hardware implementations:

Contributions in algorithmic domain:

Parallelism exploration of the architectures for turbo demodulation and turbo equalization:

- Classification of available parallelism techniques.
- Extension of the idea of sub-blocking to all components of the system
- Proposition of shuffled turbo demodulation and turbo equalization

Contributions on Hardware Implementations:

Design of two Application-Specific Instruction-set Processor (ASIP) for equalization and demapping functions:

- Analysis of flexibility need in MIMO equalization and demapping functions.
- Proposal for ASIP architectures for MIMO equalizer and soft demapper.
- Individual FPGA prototyping and validation of the two proposed ASIPs.

Design of heterogeneous multi-ASIP and Network on Chip (NoC) based high-throughput flexible and scalable turbo receiver:

- Design and FPGA prototyping of flexible transmitter.
- FPGA prototyping of a heterogeneous multi-ASIP and NoC based flexible platform capable of implementing shuffled turbo decoding, shuffled turbo demodulation and shuffled turbo equalization. The demonstration platform includes:
  - a total of 9 ASIP of three different types.
  - a total of 3 NoC instances.

Thesis Breakdown

This thesis report is divided into six chapters as follows:

Chapter 1 is dedicated to present the global requirements of future wireless terminal in terms of parameters associated with each component of the transmitter and required throughputs. To extract these requirements emerging wireless communication standards are reviewed and transmission parameters are summarized for each component of the system. Besides laying down requirements, three turbo principles, turbo decoding, turbo demodulation and turbo equalization are presented which provide promising solution in achieving error rate performance close to theoretical limits under different modes of transmission.

In the first part of the Chapter 2, a detail of Soft In Soft Out (SISO) algorithms for decoding, demodulation and equalization is presented. To address the high latency and low throughput problems of iterative processing, the available parallelism in these processes is investigated at different levels. By means of a software modeling of the proposed parallelism techniques, simulation results of parallel turbo demodulation and parallel turbo equalization are presented. Using the obtained simulation results expressions are deduced to obtain overall speed gain, area overhead and parallelism efficiency of the proposed parallel systems.

Chapter 3 is dedicated to illustrate our motivation towards the multi-ASIP and NoC based approach to achieve a flexible and scalable platform for turbo receiver. The chapter starts with an introduction to ASIP methodology to conceive flexible processing units required in a parametrized radio platform. Then, NoC paradigm is discussed to address the communication needs between multiple ASIP implementing a turbo receiver. Finally the target heterogeneous multi-ASIP and NoC based turbo receiver architecture model is introduced.

Chapter 4 presents the proposed ASIP architecture dedicated to implement MMSE-IC linear equalization algorithm for MIMO application, namely EquASIP. Based on the requirements laid down in chapter 1 for MIMO STC, the flexibility parameters are obtained for EquASIP in the start of the chapter. It is shown that how these flexibility parameters can be mapped on different basic hardware components. Based on these basic hardware building blocks, the complete ASIP architecture is presented in detail. Finally EquASIP's performance, synthesis results and its comparison with state of the art implementations are discussed.

Chapter 5 details the proposed ASIP dedicated for demapping function, namely DemASIP. Similar to chapter 4, the starting part discusses basic hardware building blocks which can be used to achieve a flexible demapper. Complete architecture of DemASIP is then presented along with the details of its functionalities. Finally DemASIP's performance, synthesis results and comparison with state of the art implementations are presented.

Chapter 6 explains the overall design and prototyping flow of the proposed heterogeneous multi-ASIP and NoC based turbo receiver. In the start, ASIP implementation flow from LISA ADL to on-board FPGA prototyping and its validation process are described. Later on step wise explanation is provided about the FPGA platform which implements flexible transmitter, channel emulator and a turbo receiver made up of 9 ASIP of three different types and 3 NoC instances.

CHAPTER

1 Multi Wireless Standard Requirements and Turbo Reception

THIS chapter presents an overall picture of a modern wireless communication system used for diverse applications such as mobile cellular network, local and wide area networks and digital video broadcasting. First of all different channel models and their effects on the transmitted data are described. A detailed explanation of transmitter components such as channel encoder, BICM interleaver, constellation mapper and MIMO STC is provided. An attempt is made to tabulate the parameters related to each of these components as suggested in different wireless standards. Finally, on the receiver side, three concepts namely turbo decoding, turbo demodulation and turbo equalization are explained in accordance with transmitter components. The last part of the chapter is dedicated to introduce a general model of an iterative receiver.

1.1 Wireless Communication System

A complete digital wireless communication system can be visualized as made up of three components namely transmitter, channel and receiver. The construction of transmitter depends on the channel model under consideration which reflects the transmission environment. Depending upon the channel, redundancy and/or diversity is added into the source data to combat against destructive effects of channel. On the receiver side the received distorted data, composed of source and redundancy, is processed to retrieve the original source. Before going deep into transmitter and receiver components, different channel models will be explained in the next section of this chapter.

1.2 Channel Models

A wireless channel is typically modeled with additive noise and multiplicative fading. The noise is added to the received signal at the input of the receiver whereas the fading influences the transmitted signal while passing through the channel. In this thesis, the additive noise is considered as white Gaussian while a fading coefficient has a Rayleigh distribution. In addition to these two main factors there are other parameters which are used to model the channel. One of the them is the presence of multi path delays comparable to the time delay between two transmitted symbols. This situation gives rise to Inter Symbol Interference (ISI) and the channel is called frequency selective. A second parameter used in characterizing the channel is the variation of the channel in time, also referred as selectivity in time.

1.2.1 Frequency Selectivity of a Channel

A channel is called frequency selective when its frequency response is not perfectly flat because of echoes and reflections generated during the transmission. This causes the transmitted signal to take paths with different attenuations and delays. The transmitted signal is then dispersed in time and the received signal includes the ISI. For a single antenna transmission system, this channel can be described by the equation:

$$y(n) = \sum_{l=0}^{L-1} h_l(n)x(n-l) + w(n) \tag{1.1}$$

where y and x represent received and transmitted signal respectively and w is Additive White Gaussian Noise (AWGN). L is the number of paths taken by the transmitted signal, reflecting the temporal dispersion of the channel during symbol transmission period. h_l represents the fading of the path l applied to a signal transmitted at time $n - l$. In this thesis we will consider the channel as frequency non-selective or flat in nature both for single antenna and MIMO systems. This is due to the reason that emerging wireless standards use OFDM technique to avoid ISI caused by frequency selectivity of the channel. Hence foe single antenna case (1.1) becomes:

$$y(n) = h(n)x(n) + w(n) \tag{1.2}$$

Similarly for a MIMO system with n_t transmit antenna and n_r receive antenna the relation between channel, transmitted symbols and received symbols is given by:

$$\mathbf{y} = \mathbf{H}.\mathbf{x} + \mathbf{w} \tag{1.3}$$

where

$$\mathbf{y} = [y_1, \ldots, y_{n_r}]^T \in \mathbb{C}^{n_r \times 1}$$
$$\mathbf{x} = [x_1, \ldots, x_{n_t}]^T \in \mathbb{C}^{n_t \times 1}$$
$$\mathbf{w} = [\omega_1, \ldots, \omega_{n_r}]^T \in \mathbb{C}^{n_r \times 1}$$
$$\mathbf{H} = \begin{bmatrix} h_{11} & \cdots & h_{1n_t} \\ \vdots & \ddots & \vdots \\ h_{n_r1} & \cdots & h_{n_rn_t} \end{bmatrix}$$

where $\mathbf{y}$ and $\mathbf{x}$ represent the received and transmitted symbol vectors respectively, $\mathbf{w}$ represents the AWGN vector and $\mathbf{H}$ is the channel matrix whose element h_{ij} represents the fading that characterizes the relation between the i^{th} receive antenna and j^{th} transmit antenna.

1.2.2 Time Selectivity of a Channel

The time selectivity characteristic of a channel define the variation of the channel with respect to time. It is related to the mobility of the transmitter, receiver or the obstacles between the two depending on the nature of fading. This selectivity characterizes 3 types of channels:

- The *fast fading channel*, which varies at each symbol period.
- The *quasi-static channel*, which remains constant during the transmission of a frame.
- The *block fading channel*, which remains constant during transmission of a given number of sub-blocks of the frame. The quasi-static channel is a special case of this type of channel.

1.3 Transmitter

In a transmitter different components are linked together to provide the immunity against channel effects and to optimally use the available channel bandwidth. A typical transmitter model with different components and their associated parameters is shown in Fig.1.1. A channel encoder which can be of a certain type provides robustness against AWGN. Similarly a bit interleaver of BICM provides protection against burst noise which destroys multiple code words in a frame. The mapper maps a combination of m bits on a complex plain containing 2^m symbols. After mapping, single antenna or MIMO transmissions are possible. In single antenna transmission SSD can be adopted against the fading effects. Whereas in MIMO an STC can be used which provides different features such as time diversity and/or spatial multiplexing. Another technique, called OFDM is used against multi path fading and used to counter the Inter Symbol Interference (ISI). This technique is, however, out of the scope of this thesis. The following subsections detail each component of a transmitter of Fig.1.1.

1.3.1 Channel Coding

The channel or error control coding is used to transmit information with maximum reliability. The principle is to introduce a redundancy in the message to enable the receiver to detect and correct the transmission errors. The coding theory, introduced by Shannon in 1948 [1], associates a channel with its capacity represented by the maximum information that can be transmitted (expressed in bits

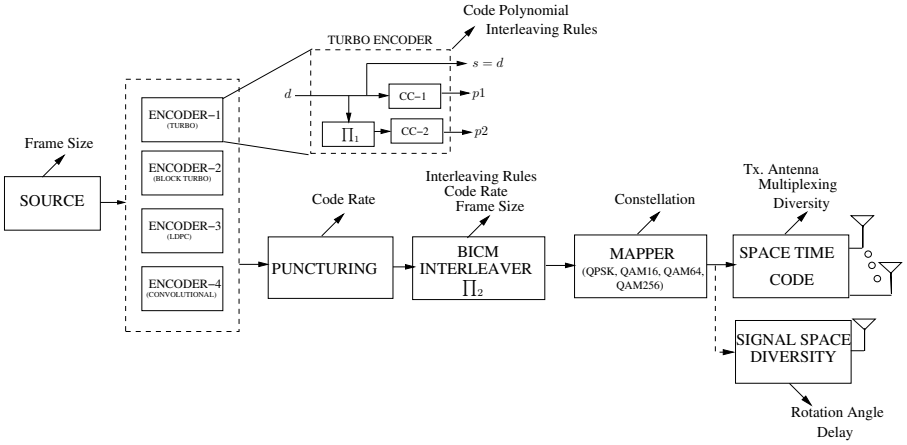


Figure 1.1 — Transmitter functional block diagram

per second) over the channel. The theorem by Shannon can be stated as: If the Information flow at the input of a channel is less than the capacity, then it is possible to transmit a digital message with an error probability arbitrarily small. In his proof, the theorem guarantees the existence of a code (the random code) for reliable transmission, but in practice the code is too complex to decode. Since then the scientific community is trying to find error correcting codes of finite length with reasonable complexity and approaching as close as possible to the channel capacity. Different codes are suggested in communication standards such as simple convolutional codes, turbo codes, block turbo codes and Low Density Parity Check codes. Following is the detail about simple convolutional encoder and turbo encoder.

1.3.1.1 Convolutional Code

A convolutional code of rate $r = \frac{n}{l}$ is a linear function which, at every instant i, transforms an input symbol d_i of n bits into an output coded symbol c_i of l bits $(l > n)$. A code is called systematic if a part of the output is made up of systematic bits $s_i = d_i$ and the rest of the bits $(l - n)$ are made up of parity.

The structure of the convolutional code is constructed with shift registers (made up of ν flip flops) and the modulo two adders (XOR). A code is characterized by a parameter called constraint length $K = \nu + 1$ where value of all ν represents one of the 2^ν states of the encoder. The code can also be called recursive if there is a feedback to the shift registers.

A convolutional code can be expressed by the generation polynomials which represents the connections between output of the registers and the modulo two adders. The Fig.1.2(a) represents a single binary non-systematic, non-recursive, 64-state encoder whose generation polynomials are: $g_1(x) = 1 + x^2 + x^3 + x^5 + x^6$ and $g_2(x) = 1 + x + x^2 + x^3 + x^7$. These generation polynomials can be represented by their coefficients, (1011011) and (1111001) respectively in binary or $(133; 171)_8$ in octal. The encoder shown in Fig.1.2(b) is a single binary Recursive Systematic Convolutional (RSC) code whereas the one shown in Fig.1.2(c) is double binary RSC code.

Trellis Representation: Although a convolutional code can be represented graphically in many ways but the trellis representation is the most popular one. The trellis diagram is made up of nodes and branches where a node represents the state s of the code and a branch represents a transition from

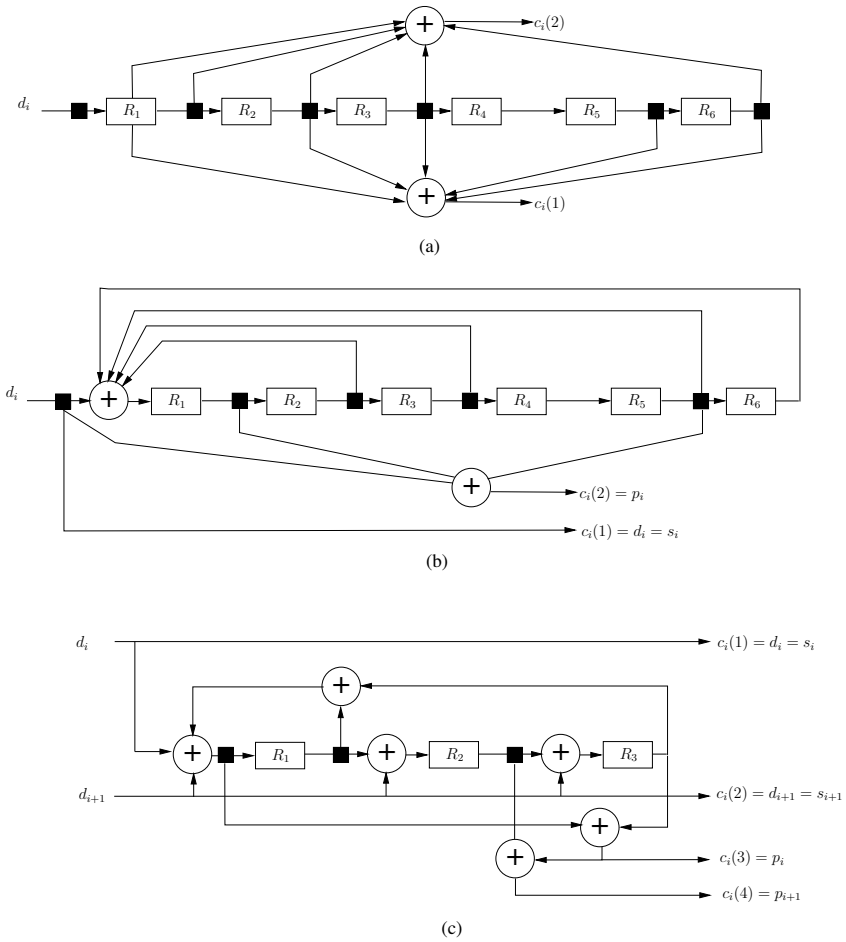


Figure 1.2 — Encoder (a) 64-state single binary code (b) 64-state single binary RSC (c) 8-state double binary RSC

one state (s) to another state (s') due to an input d. In a trellis diagram of a convoluional code, as shown in Fig.1.3, there are $2^{\nu} \times 2^{n}$ transitions, each associated to input and output vector of the encoder. A sequence of branches connecting different states makes a path and a coded frame corresponds to a unique path in the trellis.

1.3.1.2 Convolutional Turbo Code

It was known since a long time how to create codes with a correction power ensuring reliable transmission for most applications. However, this knowledge could not be turned into implementation due to the prohibitive complexity of decoders which are associated to such codes. Now, when a problem is too complex, the approach of "divide and conquer" can be a good way to simplify it. Based on this approach, the concatenation of codes has been proposed. The idea, introduced by [2], is to build a code with a sufficient correction power from several simple codes. This section briefly introduces the different proposed concatenation techniques of convolutional codes.

Concatenation of Codes: In the first architecture Forney concatenated the internal code to an external code as shown in Fig.1.4(a). This is called serial concatenation where output of outer code in input of the inner code. Subsequently, it was observed that the addition of a function of interleaving

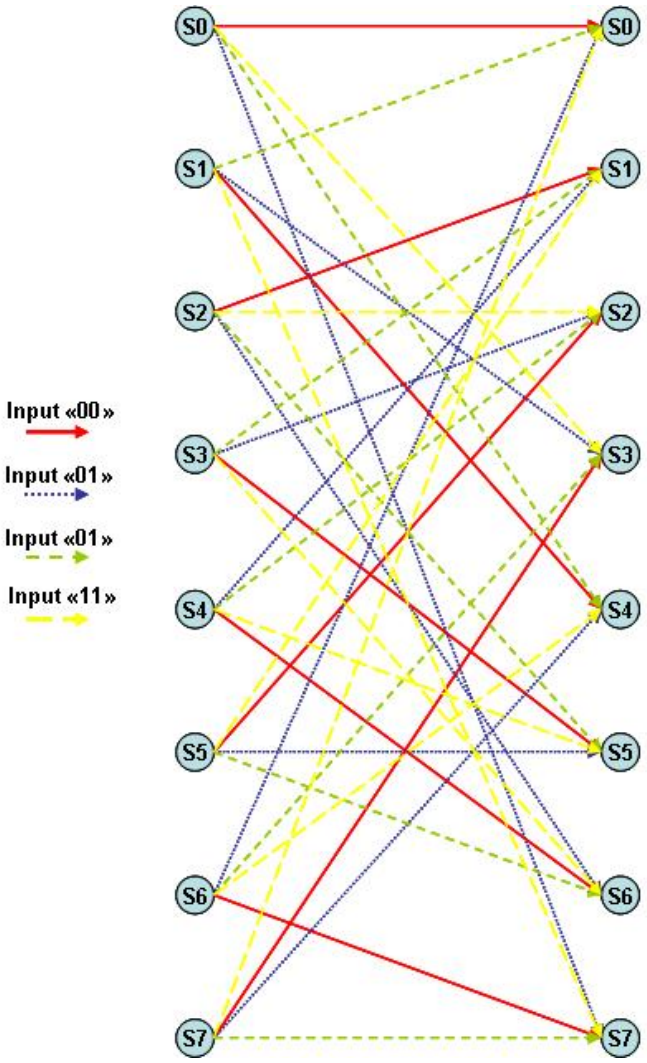


Figure 1.3 — Trellis diagram of encoder of Fig.1.2(c)

between the two codes will increase significantly robustness of the concatenated codes. Therefore what is called nowadays a serial concatenated code is more like a representation of Fig.1.4(b). With the advent of turbo codes [3], a new structure was introduced: the parallel concatenation presented in Fig.1.4(c). This structure is associated to systematic encoders where the first encoder receives the source data d in natural order and at the same time the second encoder receives the interleaved one. The output is composed of source data and associated parities in natural and interleaved domains. In this way at one instant of time parity of two different symbols are transmitted.

Turbo Code Interleaver ($\prod_1$): The interleaver in a digital communication system are used to temporally disperse the data. The primary interest of its use in concatenated codes is to put two copies of same symbol (coming to two encoders) at different interval of time. This enables to retrieve at least one copy of a symbol in a situation where the other one has been destroyed by the channel. An interleaver ($\prod$) satisfying this property can be verified by studying the dispersion factor S given by the minimum distance between two symbols in natural order and interleaved order:

$$S = \min_{i,j} \left(|i - j| + |\Pi(i) - \Pi(j)| \right) \tag{1.4}$$

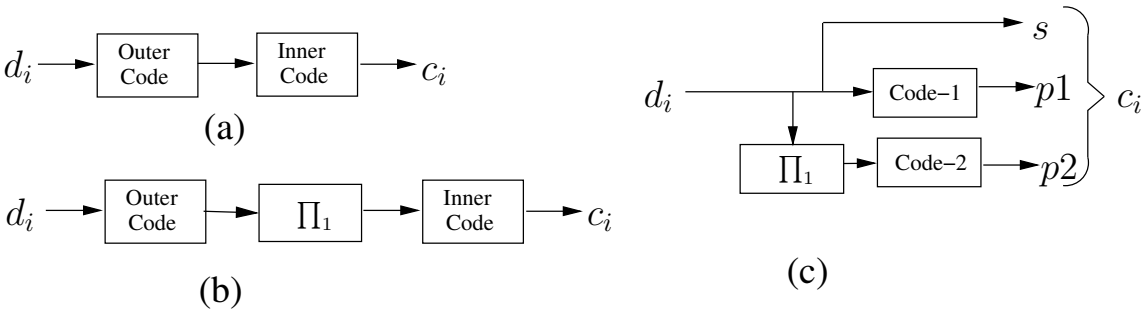


Figure 1.4 — Turbo encoder

The design of interleavers respecting a dispersion factor can be reasonably achieved through the S-random algorithm proposed in [4]. However, even if this kind of interleaver can be sufficient to validate the performance in the convergence zone of a code, it does not achieve a good asymptotic performance. Therefore to improve the latter, the design of the interleaver must also take into account the nature of component encoders. Complexity of the hardware implementation should, in addition, be taken into account. In fact, the recent wireless standards specify performance and hardware aware interleavling laws for each supported frame length.

1.3.1.3 Multi Standard Channel Coding Parameters

Multiple type of ECC have been proposed in different standards. The parameters associated to convolutional and turbo codes are detailed in the Table 1.1 for few selected standards. In following paragraphs the interleaving functions associated to turbo codes in different standards have been described:

IEEE 802.16e (WiMax)/DVB-RCS: In both of these standards, using double binary turbo code, two levels of interleaving is proposed.

1. The first one is the bit swapping in the alternate couples i.e $(d_{2j}, d_{2j+1}) = (d_{2j+1}, d_{2j})$ if $j \; mod \; 2 = 0$ where $j = 0, 1, ...N-1$ and N is number of couples in the frame.

2. The second one is given by the following expression:

$$\prod\nolimits_1(j) = (P_0 \times j + P + 1) \; mod \; N$$

where

$$\begin{aligned} P &= 0 && \text{if } j \; mod \; 4 = 0 \\ P &= \frac{N}{2} + P_1 && \text{if } j \; mod \; 4 = 1 \\ P &= P_2 && \text{if } j \; mod \; 4 = 2 \\ P &= \frac{N}{2} + P_3 && \text{if } j \; mod \; 4 = 3 \end{aligned} \tag{1.5}$$

Values of parameters P_0, P_1, P_2, P_3 depend upon the frame size and can be found in the corresponding standard specification [5] [6].

Standard	**Code Type**	**Recursive Systematic**	**Constraint Length**	**Code Polynomial**	Code Rate
802.11n	Convolutional Single Binary	-	7	$c_1 = (133)_8$ $c_2 = (171)_8$	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}$
802.16e	Convolutional Single Binary	-	7	$c_1 = (133)_8$ $c_2 = (171)_8$	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}$
	Turbo Double Binary	✓	4	$1p = (13)_8$ $2p = (11)_8$ $rec = (15)_8$	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}$
DVB-RCS	Turbo Double Binary	✓	4	$1p = (13)_8$ $2p = (11)_8$ $rec = (15)_8$	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{6}{7}$
DVB-SH	Turbo Single Binary	✓	4	$1p = (15)_8$ $2p = (17)_8$ $rec = (13)_8$	$\frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{2}{3}, \frac{2}{5}, \frac{2}{7}, \frac{2}{9}$
3GPP-LTE	Convolutional Single Binary	-	7	$c_1 = (133)_8$ $c_2 = (171)_8$ $c_2 = (165)_8$	$\frac{1}{3}$
	Turbo Single Binary	✓	4	$1p = (15)_8$ $rec = (13)_8$	$\frac{1}{3}$
DVB-T	Convolutional Single Binary	-	7	$c_1 = (133)_8$ $c_2 = (171)_8$	$\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{5}{6}, \frac{7}{8}$

Table 1.1 — Multi standard convolutional and turbo code parameters

3GPP-LTE: The expression of interleaving for 3GPP-LTE is given by:

$$\prod\nolimits_1(j) = (f_1 \times j + f_1 \times j^2)\ mod\ N$$

where $j = 0, 1, ...N-1$, N is number of bits in the frame and values of f_1 and f_2 depend on N and are given in [7].

1.3.2 Bit Interleaved Coded Modulation -BICM

First introduced by Zehavi in [8] and later on formalized by Caire *et al.* in [9], BICM offers an improvement in error correcting performance of coded modulations over fading channel. BICM is achieved by dispersing the coded binary data before mapping them to the modulated symbols. The idea is that bits related to one encoded symbol should be dispersed on different modulated symbols. By doing this bits from different coded symbols are affected by the fading effects and hence will increase the error correction capability of the decoder on the receiver side. This concept is explained with the help of Fig.1.5. In Fig. 1.5(a), bit interleaving is not applied on the bits of coded symbols (A, B, C, D, E). While passing through the channel one of the coded symbol (C) is completely destroyed. On the other hand, in Fig. 1.5(b), due to bit interleaving the bits of coded symbols are dispersed. Hence, with the same fading no single coded symbol is fully destroyed.

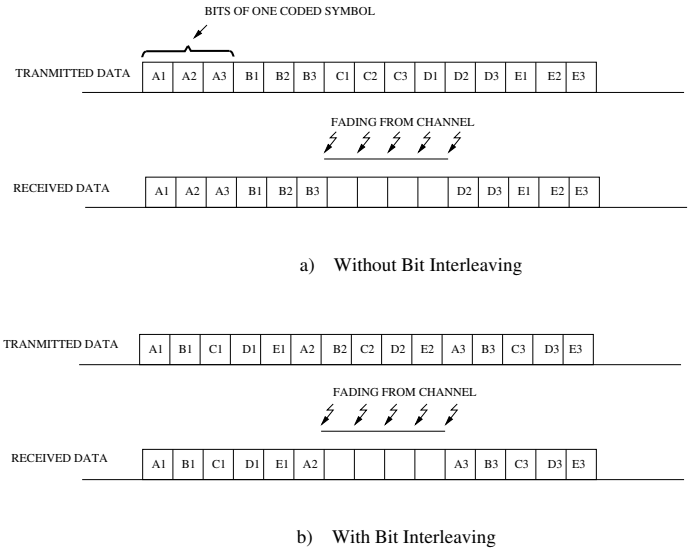


Figure 1.5 — Block of data affected by channel

The construction of the interleaver is an optimization problem taking into account issues of latency, memory requirements and the ability to disperse the noisy elements. The choice of an interleaver must be application specific and is validated by simulation.

In the following paragraphs BICM interleaver specifications for different standards are summarized.

802.11n / 802.16e: For these two standards, there is two level permutations for the interleaving. Let k be the index of the coded bits before the first permutation; i be the index after the first and before the second permutation, and j is the index after the second permutation.

1. The first permutation is given by:

$$i = \frac{N}{16} \times (k \; mod \; 16) + \lfloor \frac{k}{16} \rfloor \tag{1.6}$$

   where $k = 0, 1, ..., N$ and N is the number of bits in the coded frame.

2. The second permutation is given by:

$$j = s \times \lfloor \frac{i}{s} \rfloor + \left(i + N - \lfloor 16 \times \frac{i}{N} \rfloor \right) \; mod \; s \tag{1.7}$$

   where $s = \max(\frac{m}{2}, 1)$ and m is the number of bits per modulated symbol(concept of modulation is is explained in the next subsection).

DVB-SH/T: The interleaver used is a block interleaver and the expression is given by:

$$\prod\nolimits_2(j) = H(j) \; where \; j \; = \; 0, 1, 2, ..., N-1 \tag{1.8}$$

and N is the number of bits in the coded block. The details of $H(j)$ are summarized in Table 1.2.

Code Rate	Coded Block Size	H(j) Function
Source Block Size = 1146 Bits		
$\frac{1}{5}$	5760	$H(j) = (73 \times j)\ mod\ 5760$
Source Block Size = 12282 Bits		
$\frac{1}{5}$	61440	$H(j) = (247 \times j)\ mod\ 61440$
$\frac{2}{9}$	55296	$H(j) = (245 \times j)\ mod\ 55296$
$\frac{1}{4}$	49152	$H(j) = (221 \times j)\ mod\ 49152$
$\frac{2}{7}$	43008	$H(j) = (197 \times j)\ mod\ 43008$
$\frac{1}{3}$	36864	$H(j) = (185 \times j)\ mod\ 36864$
$\frac{2}{5}$	30720	$H(j) = (167 \times j)\ mod\ 30720$
$\frac{1}{2}$	24576	$H(j) = (157 \times j)\ mod\ 24576$
$\frac{2}{3}$	18432	$H(j) = (125 \times j)\ mod\ 18432$

Table 1.2 — BICM interleaver parameters for DVB-SH/T

3GPP-LTE: In 3GPP-LTE the bit interleaving is merged in a function called Rate Matching. After turbo encoding, systematic bits and each of the parity bits are gathered in three different sub-blocks and then interleaved separately. The sequence of interleaving, which is quite long, is available in reference [7].

1.3.3 Modulation/Mapping

The modulation process in a digital communication system maps a sequence of binary data onto a set of corresponding signal waveforms. These waveforms may differ in either amplitude or phase or in frequency, or some combination of two or more signal parameters.

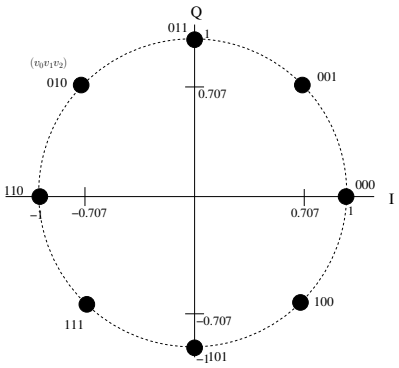


Figure 1.6 — 8-PSK constellation

1.3.3.1 Phase Shift Keying (PSK)

In this type of modulation the phase of the carrier signal is changed in accordance with the incoming sequence of the binary data. If the phase of a carrier represent m bits, then $M = 2^m$ different phases

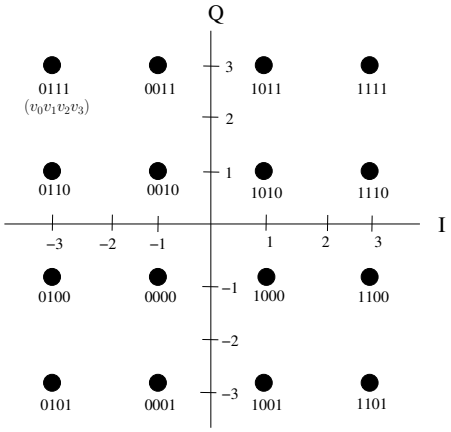


Figure 1.7 — Example of Gray mapped 16-QAM constellation

are required for all possible combinations of m bits. Mathematically this procedure can be shown by the following equation:

$$S(t) = Acos[2\pi f_c t + \frac{2\pi}{M}(i-1)] \tag{1.9}$$

where A is the magnitude of the carrier. Equation 1.9 can be expanded as:

$$S(t) = Acos\frac{2\pi}{M}(i-1)cos2\pi f_c t - Asin\frac{2\pi}{M}(i-1)sin2\pi f_c t \tag{1.10}$$

where $i = 1, 2, 3...M$.

In the above equation the $cos2\pi f_c t$ and $sin2\pi f_c t$ are the two basic signals (90^o a part in phase) and the factors $Acos\frac{2\pi}{M}(m-1)$ and $Asin\frac{2\pi}{M}(m-1)$ represent the multiplicative coefficients of these signals respectively. If we set the value of M = 2, 4, 8 then the corresponding PSK is called BPSK, QPSK and 8-PSK. The other representation of the above equation is in the form of Signal Space Diagram (constellation Diagram) where $cos2\pi f_c t$ and $sin2\pi f_c t$ can be visualized as in-phase (I) and Quadrature (Q) axis of complex plane respectively.

Gray coding is used in constellation mapping as it gives two advantages. First, if the detection is symbol based, the error caused by receiving the adjacent symbol as compared to transmitted symbol will be of one bit. Second, use of Gray coding provides symmetries in the constellation. The Gray coding and the respective I and Q components for 8-PSK are shown in the Figure 1.6.

1.3.3.2 Quadrature Amplitude Modulation (QAM)

In this type of modulation the amplitude of two carriers (90^o a part in phase) are changed in accordance with the incoming sequence of the digital data:

$$S(t) = A_c cos2\pi f_c t - A_s sin2\pi f_c t \tag{1.11}$$

Here again we use the same two carrier signals as basic signals and A_c and A_s represent the coefficients of these two signals depending upon the different symbols in the modulation scheme. The constellation diagram of Gray mapped 16-QAM modulation is shown in Fig. 1.7.

1.3.3.3 Multi Standard Mapper Specifications

In Table 1.3, parameters related to mapping function adopted in different standards have been summarized.

Standard	Modulation Type Support
IEEE-802.16e	QPSK, 16-QAM and 64-QAM
IEEE-802.11	BPSK, QPSK, 16-QAM and 64-QAM
3GPP-LTE	QPSK, 16-QAM and 64-QAM
DVB-SH	QPSK, 8PSK and 16APSK
DVB-S2	QPSK, 8PSK, 16APSK and 32APSK
DVB-T	QPSK, 16-QAM and 64-QAM
DVB-T2	QPSK, 16-QAM, 64-QAM and 256-QAM

Table 1.3 — Modulation types supported in different standards

1.3.4 Signal Space Diversity-SSD

SSD is a way of adding diversity in a modulated symbol before the transmission. Two low complexity solutions have been proposed in [10] to double the diversity order of Turbo BICM scheme (using turbo code and BICM in the transmitter). The proposed solutions are:

- correlating the in-phase I and quadrature Q component of the transmitted signal
- making these two components to fade independently

1.3.4.1 Correlating I and Q Components

When Gray mapping is used, QAM schemes are reduced to two independent Pulse Amplitude Modulations (PAM) on every component axis represented by the I and Q channels. In Fig.1.7, bits v_0 and v_1 are mapped on I channel independently of bits v_2 and v_3 which are mapped on Q channel. Hence all constellation points cannot be uniquely identified in the I channel or Q channel separately.

In order to circumvent this natural independence and hope for any improvement in the diversity order, one should correlate the I and Q channels on every constellation point. This correlation has as purpose to uniquely identify every constellation point from any component axis. Since Gray mapping provides best performance with and without Iterative Demapping (ID) when turbo code is used [11], a simple solution to correlate both channels is to rotate the constellation as shown in Fig.1.8. This rotation does not change neither the distances between constellation points nor the distance to the origin hence no modification in transmission power or bandwidth is required.

1.3.4.2 Independent Fading of I and Q Components

When a transmitted constellation point is subject to a fading event, its I and Q coordinates fade identically. When subject to a destructive fading, the information transmitted on I and Q channels suffers from an irreversible loss leading to an erroneous detection of the symbol at the receiver side. If I and Q fade independently, in most cases it is highly unlikely to have severe fading on both components. One way to allow both components to fade independently has been proposed in [12]

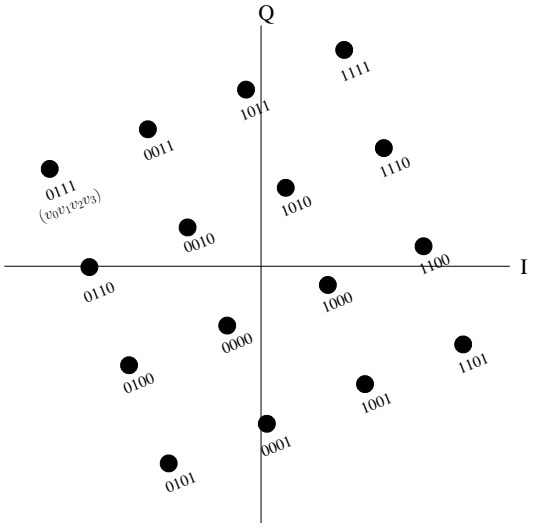


Figure 1.8 — Example of rotated Gray mapped 16-QAM

which is to interleave I and Q components. Another simpler way [13] is to introduce a time delay d between I and Q component transmission. In case of uncorrelated flat fading a delay of only one symbol period between the I component with respect to the Q component is sufficient to have these two components suffer differently from fading amplitudes.

1.3.4.3 Multi Standard SSD Specifications

This recently developed method of adding diversity in the modulated signal is now adopted in the emerging DVB-T2 standard. Different rotation angles α are given in Table 1.4.

Standard	QPSK	16-QAM	64-QAM	256-QAM
α (Degree)	29.0	16.8	8.6	3.57

Table 1.4 — SSD Parameters for DVB-T2 standard

1.3.5 MIMO Space Time Code-STC

At the transmitter, MIMO techniques are employed to exploit the diversity issue and/or to increase throughput. To do this, a space-time code should be implemented. The basic idea of space-time coding is to create redundancy or correlation between symbols transmitted on the spatial and temporal dimensions. A space-time code is characterized by its rate, the order of diversity and coding gain. The rate of space-time code is equal to the ratio between the number of symbols emitted and their corresponding number of transmission periods. The order of diversity is the number of independent channels at the reception. Finally, the coding gain is the gain made by the coded system, in terms of performance, compared to non-coded system. A space-time code is said to be full rate when the rate is equal to the number of antennas at the transmitter. A space-time code is said to have maximum diversity when it is able to exploit a diversity equal to $n_t \times n_r$.

1.3.5.1 Diversity Techniques

For the exploitation of diversity, space-time coding can be divided into two main classes : trellis and block coding.

Space Time Trellis Code-STTC: In this technique, the symbols to be transmitted on different antennas are encoded using a representation of a trellis (state machine). The decoding is done typically by the Viterbi algorithm by minimizing a metric of cumulative probability to choose the most likely path in the trellis. It is shown in [14] that trellis codes can operate at maximum diversity transmission and reception while providing a coding gain that depends on the number of states of the trellis. On the other hand, the decoding complexity increases exponentially with the number of transmit antennas and modulation order. The complexity of the decoder precludes the implementation of this technique.

Space Time Block Code-STBC: The complexity of implementation of STTC codes has motivated the construction of space-time block code. They are defined in matrix form. The symbols to be transmitted is encoded by matrix operations. In [15], Alamouti has built a space-time orthogonal reaching the maximum diversity for a MIMO system 2×1 with a rate equals to 1 (equivalent to the performance of a single antenna system). The code in matrix form is given as:

$$A = \begin{bmatrix} x_i & -x_{i+1}^* \\ x_{i+1} & x_i^* \end{bmatrix}$$

where columns in matrix A show the time and rows of the matrix indicate the data on two transmit antennas. $(.)^*$ represents the complex conjugate operation. In [16], Tarokh generalized Alamouti code to higher dimensions. The advantage of these orthogonal codes is their linear decoding. Moreover, they can achieve maximum diversity but their rate is limited to 1. This constraint has motivated the construction of space-time codes, called quasi-orthogonal, to achieve rates greater than 1. Another family of STBC codes, known as linear dispersion [17] is generically obtained from linear combinations of symbols to be transmitted. Among the codes of this family, we can cite the Golden code [18] representing a perfect code for a 2×2 system and is shown below as C matrix.

$$C = \frac{1}{\sqrt{1+r^2}} \begin{bmatrix} x_i + jr.x_{i+3} & r.x_{i+1} + x_{i+2} \\ x_{i+1} - r.x_{i+2} & jr.x_i + x_{i+3} \end{bmatrix}$$

where $r = \dfrac{-1+\sqrt{5}}{2}$.

1.3.5.2 Multiplexing Techniques

The limited rate of orthogonal codes has motivated the construction of codes in layers with a full rate feature. Foschi [19] offers a first scheme called Bell Laboratories Layered Space Time (BLAST) exploiting the spatio-temporal multiplexing in a multi-antenna system. The binary transmit frame is divided into sub-frames. The sub-frames (layers) are then transmitted on different antennas along a vertical (V-BLAST), horizontal (H-BLAST) or diagonal (D-BLAST) distribution. When an STC is used in combination with channel coding and BICM the scheme is called ST-BICM.

1.3.5.3 ST-BICM

At emitter side, a MIMO transmission with channel coding consists of dividing a frame of binary data to sub-frames. The sub-frames are encoded, interleaved and modulated by one or more layers. They are then transmitted on different antennas along a vertical, horizontal or diagonal distribution. From the three possible distributions, we obtain the three following techniques.

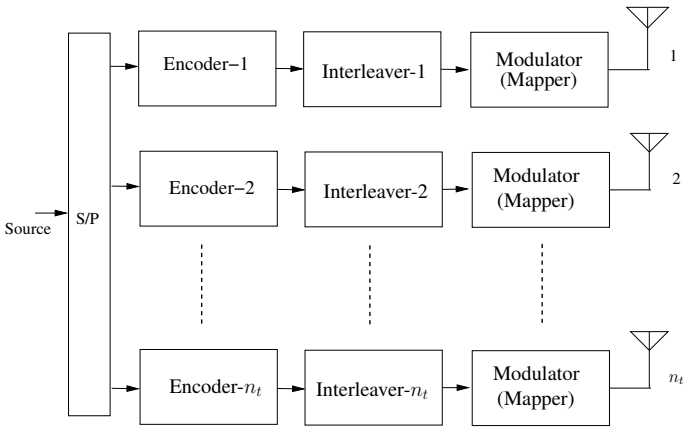


Figure 1.9 — Structure of HE-LST

Horizontal Encoding - Layered Space Time (HE-LST) : Each information sub-frame is encoded, interleaved, modulated and transmitted independently by each transmitting antenna (Fig. 1.9). The advantage of this system is that it is flexible and simple to implement, but in return it does not exploit the spatial diversity of the transmission system.

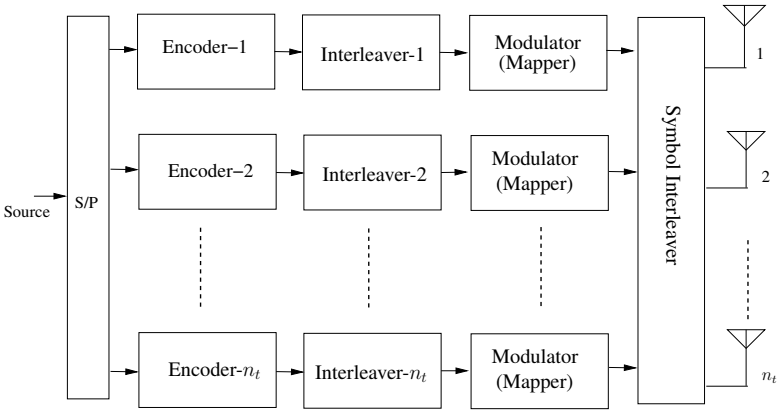


Figure 1.10 — Structure of DE-LST

Diagonal Encoding - Layered Space Time (DE-LST) : Each information sub-frame is encoded, interleaved, modulated independently (Fig. 1.10). The symbols modulated on each sub-frame are then transmitted sequentially by each antenna using a diagonal type of space-time interleaver. Thus, each transmitted sub-frame undergoes all different fadings of MIMO channel. The advantage of this system is that it exploits the transmit diversity, but its implementation remains difficult.

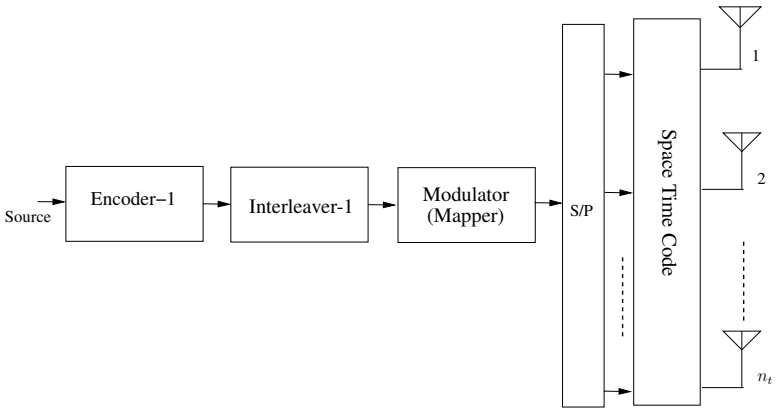


Figure 1.11 — Structure of VE-LST

Vertical Encoding - Layered Space Time (VE-LST) : The information frame is encoded, interleaved and modulated in a single layer (Fig. 1.11). Each modulated symbol of the encoded frame is then transmitted by a transmit antenna. It can be seen as BICM associated with space-time coding. One can say it as ST-BICM structure *(*Space Time BICM), an extension of BICM using multiple antennas at the transmission. This system offers a compromise between complexity and diversity as firstly the frame breaking is done after modulation but before transmission, and secondly each symbol undergoes different fading in the channel.

1.3.5.4 MIMO-STC Specifications

Diversity and/or multiplexing achieved through MIMO in different standards are summarized in Table 1.5.

MIMO Feature	IEEE 802.11n	IEEE 802.16e	3GPP-LTE
Time Diversity(Alamouti)	✓	✓	✓
Spatial Multiplexing	✓	✓	✓
Golden Code		✓	
Mixed Diversity/ Multiplexing		✓	✓

Table 1.5 — Multi standard MIMO support

1.3.6 Data Rate Requirements

The data rates requirements associated to different emerging standards are summarized in Table 1.6 which is a challenging aspect for the hardware designers.

1.4 Turbo Receiver

On the receiver side the objective is to remove the channel effects to retrieve the original source data. This objective is achieved by exploiting the redundancy and diversity added to source data in the

Standard	Throughput (Mbps)	Standard	Throughput (Mbps)
802.11	11-540	DVB-RCS	2-20
802.16	100	DVB-S2	100-200
3GPP-LTE	150	DVB-T2	10-50
DVB-T	10-25	DVB-H	10

Table 1.6 — Data rate requirement in emerging wireless communication standards

transmitter. After the introduction of turbo decoding, the receiver structure can be categorized into two global categories: iterative/turbo receivers and non-iterative receivers. In a non-iterative receiver, each constituent unit processes the data once and then passes the information to the next unit. In an iterative receiver there are feedback paths in addition to forward paths, through which, constituent units can send the information to previous units iteratively. A Soft In Soft Out (SISO) processing block of an iterative receiver, using channel information and the information received from other units, generates soft outputs at each iteration. Depending upon the link of feedback path in the receiver, three iterative processes under consideration in this thesis can be categorized as turbo decoding, turbo demodulation and turbo equalization which are explained in following subsections.

1.4.1 Turbo Decoding

Iterative/Turbo decoding of turbo codes involves an exchange of information between constituent component decoders. This exchange is enabled by linking the *a priori* soft probability input port of one component decoder to the extrinsic soft probability output port provided by the other component decoder (Fig.1.12). Only the extrinsic information is transmitted to the other decoder, in order to avoid the information produced by the one decoder being fed back to itself. This constitutes the basic principle of iterative decoding. The decoding principle of Parallel Concatenation Convolutional Codes

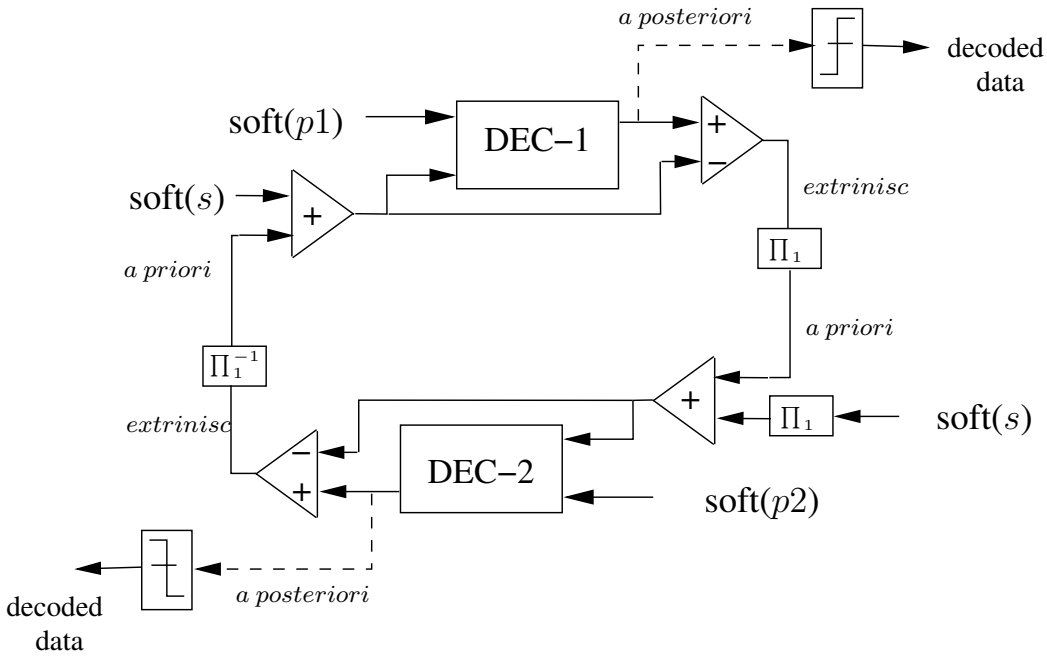


Figure 1.12 — Parallel concatenated convolutional turbo decoder

(PCCC) and Serially Concatenation Convolutional Codes (SCCC) is shown in Fig.1.12 & 1.13. The SISO decoders are assumed to process soft values of transmitted bits at their inputs. In the PCCC scheme of Fig.1.12, each SISO decoder computes the extrinsic information related to information symbols, using the observation of the associated systematic and parity symbols coming from the transmission channel and the *a priori* information. Since no *a priori* information is available from the decoding process at the beginning of the iterations they are not used. For the subsequent iterations, the extrinsic information coming from the other decoder are used as *a priori* information for the current SISO decoder. The decisions can be computed from any of the decoders. In the original parallel concatenation case, the turbo decoder structure is symmetrical with respect to both constituent decoders. However, in practice, the SISO processes are executed in a sequential fashion. The decoding process starts arbitrarily with either one decoder, DEC-1 for example. After DEC-1 processing is completed, DEC-2 starts processing and so on.

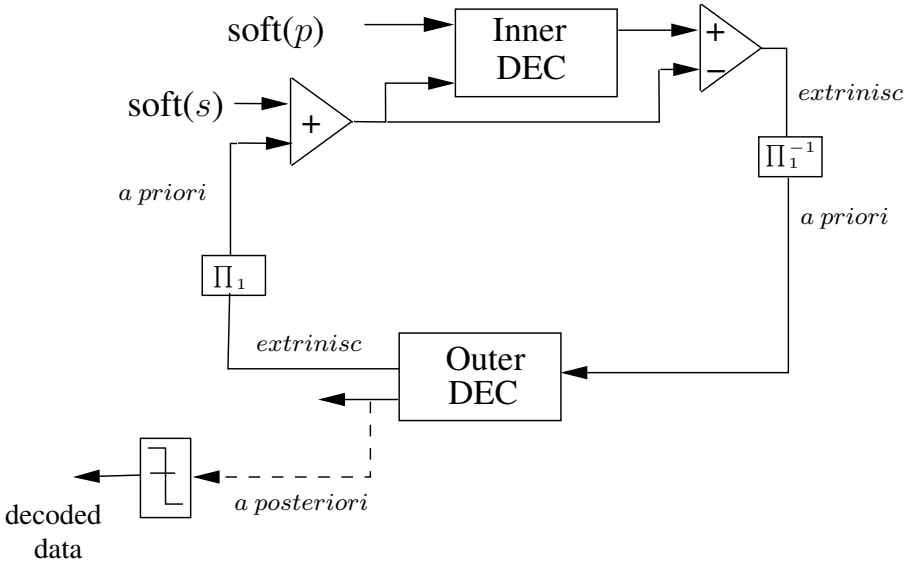


Figure 1.13 — Serially concatenated convolutional turbo Decoder

In the SCCC scheme of Fig.1.13, the decoding diagram is no longer symmetrical. On one hand, the inner SISO decoder computes extrinsic information related to the inner code information symbol, using the observation of the associated coded symbols coming from the transmission channel and the extrinsic information coming from the other SISO decoder. On the other hand, the outer SISO decoder computes the extrinsic related to the outer code symbols using the extrinsic information provided by the inner decoder. The decisions are computed as *a posteriori* information related to information symbols by the outer SISO decoder. Although the overall decoding principle depends on the type of concatenation, both turbo decoders can be constructed from the same basic building SISO blocks. Each SISO decoder processes its own data and passes it to the other SISO decoder. One iteration corresponds to one pass through each of all the SISO decoders. One pass through a single SISO decoder is sometimes referred to as half an iteration of decoding.

1.4.2 Turbo Demodulation

When the transmission is coded and turbo demodulation or iterative demapping is adopted, extrinsic information at the output of the decoder is fed back as *a priori* soft information to the input of the demapper or demodulator. The Bit Error Rate (BER) performance at the output of the demod-

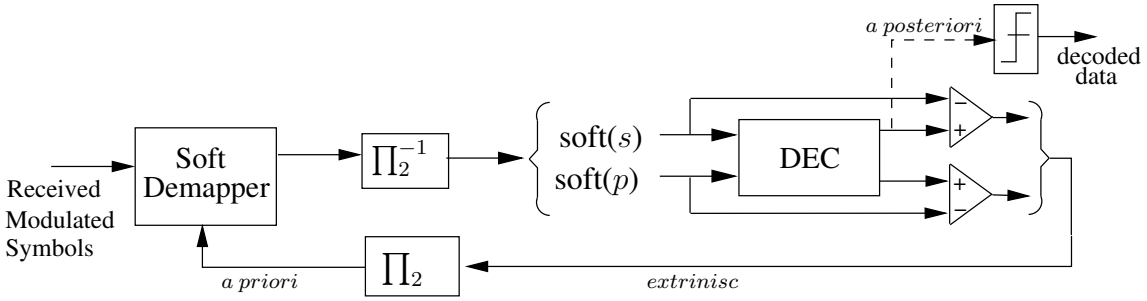


Figure 1.14 — Iterative demapping for convolution code

ulator/demapper in the case of iterative demodulation tends to the one of the genie-aided demodulator/demapper as the Signal to Noise Ratio (SNR) increases [10]. For a system with convolutional code, BICM and 8PSK, iterative demapping was studied in [20] both for Rayleigh fading and AWGN channels. The scheme is shown in Fig.1.14. One dB and 1.5dB gains were reported in this paper for Rayleigh and AWGN channels respectively. In [13] authors studied the effects of mapping style of 16-QAM and independent fading of I and Q Component application on iterative demapping (ID) for Rayleigh fading channel. A gain of only 0.1dB was reported in [21] when convolutional code was

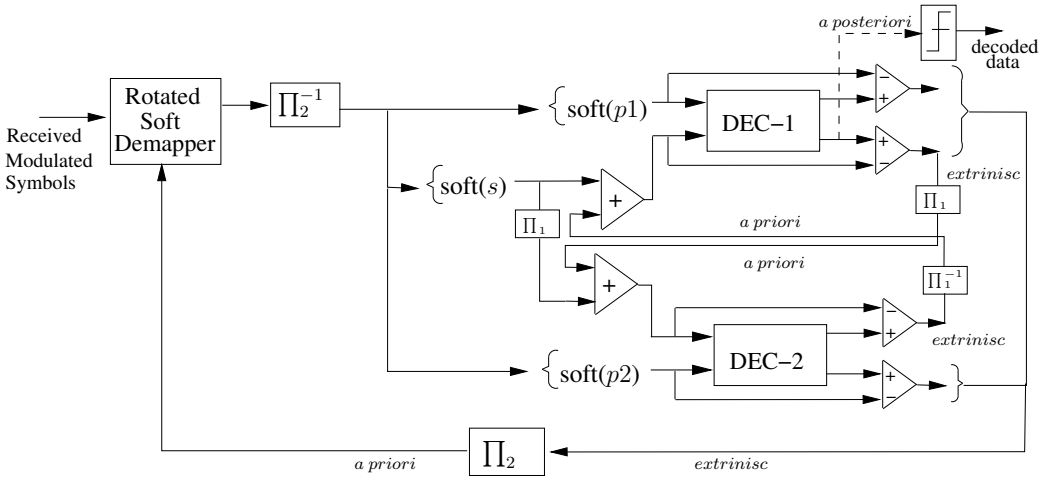


Figure 1.15 — Iterative demapping for convolution turbo code

replaced with turbo code. This result makes BICM-ID with turbo-like coding solutions unsatisfactory even though the added complexity is relatively small. In [22], authors proposed a turbo BICM-ID approach designed for fading channels (Fig.1.15). This approach offers significant performance gains, especially for a BER lower than 10^{-7} , over existing classical turbo BICM systems at the cost of a small increase in system complexity. The proposed iterative demapper takes advantage of the introduced SSD to significantly lower the error floor without sacrificing iterative process convergence.

1.4.3 Turbo Equalization

The traditional methods of data protection used in error correction code do not work well when the channel over which the data is sent introduces additional distortions in the form of ISI. When the channel is frequency selective or for other reasons is dispersive in nature, the receiver will need to

compensate for the channel effects prior to employing a standard decoding algorithm for the error correction code. Such methods for channel compensation are typically referred to as channel equalization. Similarly to turbo demodulation, the soft information from the decoder can be properly interleaved and taken into account in the equalization process, creating a feedback loop between the equalizer and decoder. Through this link each of the constituent algorithms communicates its beliefs about the relative likelihood that each given bit takes on a particular value. This concept, known as turbo equalization, was first introduced in [23] to combat the detrimental effects of ISI for digital transmission protected by convolutional code. For coded BICM system the architecture of turbo equalization is shown in Fig.1.16. For the first time the equalizer removes the effects of ISI and the

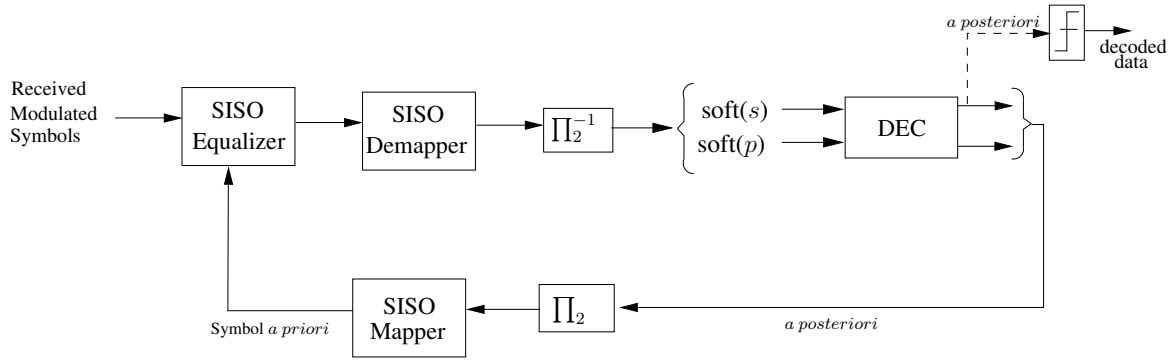


Figure 1.16 — Turbo equalization for convolution code

SISO demapper outputs information related to bits which are used by decoder which generates the *a posteriori* information both on systematic and parity bits. This information is fed back both to SISO equalizer. Since the equalizer is working on symbols, the *a posteriori* information on bits is used in SISO mapper to create the symbols which serve as *a priori* for the equalizer.

In the emerging wireless standards where MIMO has been inducted, in addition to ISI, there is co-antenna interference at the MIMO reception. To address the effects of ISI related to frequency selectivity of the channel, OFDM is generally used in advanced communication applications. In a MIMO-OFDM system, where a receiver should combat against the co-antenna interference, the concept of turbo equalization can be used to cancel the interference caused by MIMO iteratively as shown in Fig.1.17.

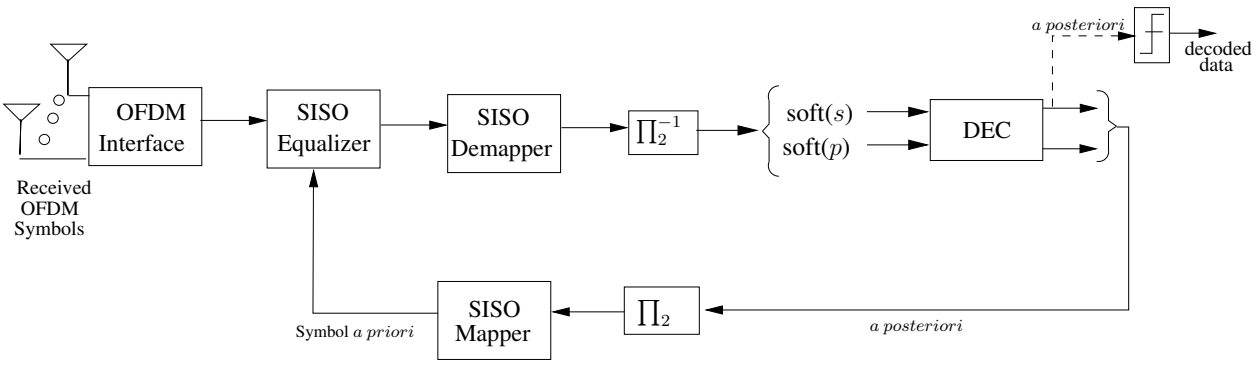


Figure 1.17 — Turbo equalization in MIMO-OFDM

1.4.4 Unified Turbo Receiver

A unified turbo receiver block diagram is proposed in Fig. 1.18. In the proposed architecture, the three above-described iterative concepts are combined. Depending upon the requirements imposed by the

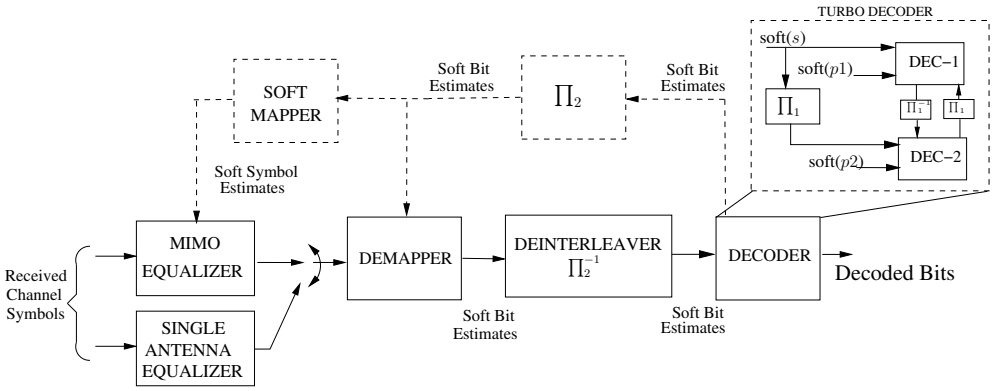


Figure 1.18 — Unified turbo receiver

nature of the channel and the target error rate performance, the feedback links can be used or not to achieve desired performance from the proposed unified architecture.

1.5 Conclusion

In this first chapter we laid down the basic requirements of an advanced wireless digital communication system by summarizing various parameters associated with the transmitter, the channel and the receiver. Regarding the channel, single and multiple antenna communication channels were categorized on the bases of frequency and time selectivity. Different components of the transmitter such as channel encoder, BICM interleaver, mapper and MIMO-STC were explained. A summary of the parameters related to each of these components, adopted in different emerging standards, has also been tabulated together with data rate requirements. In the last part of this chapter, an effort has been made to briefly describe the principles of three turbo processes namely turbo decoding, turbo demodulation and turbo equalization. These three iterative concepts have been, finally, combined to provide a unified turbo receiver architecture which represents the target of this thesis work.

CHAPTER

2 Turbo Reception Algorithms and Parallelism

IN the previous chapter stringent requirements, primarily flexibility and high throughput, imposed by upcoming wireless standards for different applications have been discussed. The other discussed point was the introduction of advanced techniques such as turbo codes, SSD and MIMO which encourages the adoption of iterative processing such as turbo decoding, turbo demodulation and turbo equalization in the receiver to acquire error rate performances near theoretical limits. However, the conventional serial execution of these iterative processes constitutes a major issue to the throughput requirement and thus prohibits their wide use.

To address this issue, parallelism opportunities at all levels of a turbo receiver is investigated and presented in this chapter. To that end, first the algorithms used in different components of a turbo receiver are presented in the start of this chapter. The simplification applied to these algorithms to make them suitable for hardware implementation are also explained. The parallelism in these iterative algorithms is then explored and available techniques are classified in three different levels. Finally, in order to analyze the parallelism performance both for parallel turbo demodulation and parallel turbo equalization, corresponding software models are created which simulate the first and second level of parallelisms. Simulations results in terms of speed gain and parallelism efficiency are presented and analyzed for different system configurations.

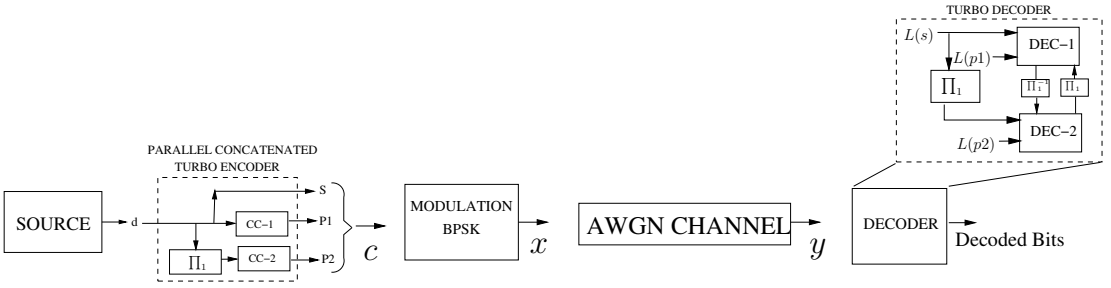


Figure 2.1 — System diagram with turbo encoder, BPSK modulator, AWGN channel and turbo decoder

2.1 Soft In Soft Out (SISO) Decoding Algorithm

Consider the system diagram of Fig.2.1 where source is encoded by convolutional parallel concatenated turbo encoder which outputs the source (systematic) and the parity bits. These encoded bits pass through a mapper which applies Binary Phase Shift Keying (BPSK) modulation to produce modulated symbol x. Then noise is added to these symbols due to AWGN channel. The received corrupted symbols y are the input to the turbo decoder. The turbo decoder is comprised of two component decoders DEC-1 and DEC-2 responsible for the error correction.

If we look at the history of decoding algorithms, several ones have been proposed to decode a convolutional code. The initial algorithms are presented by Fano [24] and Viterbi [25] which have binary inputs and outputs. The Viterbi algorithm which is better than the other was later modified to accept the soft inputs to improve the decoding [26]. The Soft Output Viterbi Algorithm (SOVA) [27] takes the soft input and provides the soft output as well. Among the SISO algorithms, the Cock-Bahl-Jelinek-Raviv (BCJR) [28] also called MAP (Maximum A Posteriori) or forward backward algorithm, is the optimal decoding algorithm which calculates the probability of each symbol from the probability of all possible paths in the trellis between initial and final states. In practice, due to its complexity, the BCJR algorithm is not implemented in its probabilistic form but rather used in simplified logarithmic domain to transform multiplications into additions. Hence these simplified versions are named as log-MAP or in sub optimal form as max-log-MAP algorithms.

2.1.1 MAP Decoding Algorithm

For each source symbol d_i comprised of n bits, encoded in l output bits by an encoder having ν memory elements (i.e 2^ν) states at rate $r = \frac{n}{l}$, a MAP decoder provides 2^n *a posteriori* probabilities with the full knowledge of the sequence y received by the decoder i.e a decision for each possible value of symbol. The hard decision is the corresponding value j that maximizes the a posteriori probability. These probabilities can be expressed in terms of joint probabilities.

$$Pr(d_i \equiv j|y) = \frac{p(d_i \equiv j, y)}{\sum_{k=0}^{2^n-1} p(d_i \equiv k, y)} \tag{2.1}$$

The trellis structure of the code allows to decompose the calculation of joint probabilities between past and future observations. This decomposition utilizes the forward recursion metric $\alpha_i(s)$ (the probability of a state of the trellis at instant i computed from past values), backward recursion metric $\beta_i(s)$ (the probability of a state of the trellis at instant i computed from future values), and a metric

$\gamma_i(s^{'}, s)$ (the probability of a transition between two states of the trellis). Using these metrics the expression of 2.1 becomes:

$$p(d_i \equiv j|y) = \sum_{(s',s)/d_i \equiv j} \beta_{i+1}(s)\alpha(s')\gamma_i(s', s) \tag{2.2}$$

The forward and backward recursion metrics are calculated in following way:

$$\alpha_{i+1}(s) = \sum_{s'=0}^{2^\nu - 1} \alpha_i(s')\gamma_i(s', s), \quad \text{for } i = 0 \ldots N-1 \tag{2.3}$$

$$\beta_i(s) = \sum_{s'=0}^{2^\nu - 1} \beta_{i+1}(s')\gamma_i(s, s'), \quad \text{for } i = N-1 \ldots 0 \tag{2.4}$$

The initialization of these metrics depends on the knowledge of initial and final state of the trellis, e.g if the encoder starts at state S_0 then $\alpha_0(S_0)$ has value 1 while other $\alpha_0(s)$ will be 0. If the initial state is unknown then all states are initialized to same equiprobable value.

Similarly the branch metric can be expressed in the following way:

$$\gamma_i(s', s) = p(y_i|x_i).\Pr^a(d_i = d_i(s', s)) \tag{2.5}$$

where $p(y_i|x_i)$ represents the channel transition probability which can be expressed for a Gaussian channel in following way:

$$p(y_i|x_i) = \prod_{k=1}^{l} \left(\frac{1}{\sigma\sqrt{2\pi}}.e^{-\frac{\left(y_{i,k} - x_{i,k}\right)^2}{\sigma^2}} \right) = K.e^{\frac{\sum_{k=1}^{l} y_{i,k}.x_{i,k}}{\sigma^2}} \tag{2.6}$$

where x_i is i^{th} transmitted modulated symbol.

The *a priori* probability $\Pr^a(d_i = d_i(s', s))$, to emit m-ary information corresponding to transition from s' to s is 0 if the transition does not exist in the trellis. Otherwise its value depends upon the statistics of the source. For an equiprobable source $Pr^a(d_i = j) = \frac{1}{2^n}$. In the context of the turbo decoding, the *a priori* probability takes into account the input extrinsic information.

The extrinsic information generated by the decoder is computed in the same way as *a posteriori* information (2.1) but with a modified branch metric:

$$Pr^{ex}(d_i \equiv j|y) = \frac{\sum_{(s',s)/d_i \equiv j} \beta_{i+1}(s)\alpha(s')\gamma_i^{ex}(s', s)}{\sum_{(s',s)} \beta_{i+1}(s)\alpha(s^{'})\gamma_i^{ex}(s', s)} \tag{2.7}$$

Hence the branch metric does not take into account the already available information of a symbol for which extrinsic information is being generated. For parallel convolutional turbo codes, systematic part is removed from the branch metric computation and can be expressed as:

$$\gamma_i^{ex}(s', s) = K.e^{\frac{\sum_{k=n+1}^{l} y_{i,k}.x_{i,k}}{\sigma^2}} \tag{2.8}$$

2.1.2 Log-MAP or max-log-MAP Decoding Algorithm

The log-MAP algorithm which is introduced by [29], is the direct transformation of MAP algorithm in logarithmic domain. Hence, all the metrics M of MAP algorithm will be replaced by metrics $\sigma^2 \ln M$. This permits to transform the semi ring sum-product $(R^+, +, \times, 0, 1)$ to semi ring $(R, max^*, +, -\infty, 0)$ where the max^* operator is defined as below:

$$max^*(x,y) = \sigma^2 \ln\left(e^{\frac{x}{\sigma^2}} + e^{\frac{y}{\sigma^2}}\right) = max(x,y) + \sigma^2 \ln\left(1 + e^{-\frac{|x-y|}{\sigma^2}}\right) \approx max(x,y) \tag{2.9}$$

This operator can be simplified by an operator max which corresponds to max-log-MAP algorithms. Using this approximation the branch metrics of (2.5) and (2.8) can be written as:

$$c_i\left(s',s\right) = \sigma^2 \ln \gamma_i\left(s',s\right) = K' + L_i^a(j) + \sum_{k=1}^{l} y_{i,k}.x_{i,k} = c_i^{ex}\left(s',s\right) + L_i^a(j) + L_i^{sys}(j) \tag{2.10}$$

and

$$c_i^{ex}\left(s',s\right) = \sigma^2 \ln \gamma_i^{ex}\left(s',s\right) = K' + \sum_{k=n+1}^{l} y_{i,k}.x_{i,k} \tag{2.11}$$

where $L_i^a(j)$ is the metric of *a priori* information and $L_i^{sys}(j)$ is the metric which corresponds to systematic part of the data. It is interesting to note that constant K' is not necessary in practice because it is removed while computing (2.14) and (2.15).

In the same way the forward and backward recursion metrics can be written as:

$$a_{i+1}(s) = \sigma^2 \ln \alpha_{i+1}(s) = \underset{s'=0}{\overset{2^v-1}{max^*}}\left(a_i\left(s'\right) + c_i\left(s',s\right)\right) \quad \text{pour } i = 0...N-1 \tag{2.12}$$

$$b_i(s) = \sigma^2 \ln \beta_i(s) = \underset{s'=0}{\overset{2^v-1}{max^*}}\left(b_{i+1}\left(s'\right) + c_i\left(s,s'\right)\right) \quad \text{pour } i = N-1...0 \tag{2.13}$$

In this case the first and last metrics are initialized in following way :

- in case state S is known, $a(S) = 0$ while the others are $a(s \neq S) = -\infty$.
- in case state is unknown, all the states will have $a(s) = 0$.

The *a posteriori* (2.1) and extrinsic (2.7) informations are transformed into:

$$\begin{aligned} L_i(j) &= \sigma^2 \ln \Pr\left(d_i \equiv j | y\right) \\ &= \underset{(s',s)/d_i \equiv j}{max^*}\left(a_i\left(s'\right) + c_i\left(s',s\right) + b_{i+1}(s)\right) - \underset{(s',s)}{max^*}\left(a_i\left(s'\right) + c_i\left(s',s\right) + b_{i+1}(s)\right) \end{aligned} \tag{2.14}$$

$$L_i^{ex}(j) = \underset{(s',s)/d_i \equiv j}{max^*}\left(a_i\left(s'\right) + c_i^{ex}\left(s',s\right) + b_{i+1}(s)\right) - \underset{(s',s)}{max^*}\left(a_i\left(s'\right) + c_i^{ex}\left(s',s\right) + b_{i+1}(s)\right) \tag{2.15}$$

By simplifying the terms on the right side of the metrics, using the most probable symbol $\widehat{j}$ in the following way, one will have:

$$\underset{(s',s)}{max^*}\left(a_i\left(s'\right) + c_i\left(s',s\right) + b_{i+1}(s)\right) = \underset{j=0}{\overset{n}{max^*}}\left(\underset{(s',s)/d_i \equiv j}{max^*}\left(a_i\left(s'\right) + c_i\left(s',s\right) + b_{i+1}(s)\right)\right) \tag{2.16}$$

$$\begin{aligned} &\approx \underset{j=0}{\overset{n}{max}}\left(\underset{(s',s)/d_i \equiv j}{max^*}\left(a_i\left(s'\right) + c_i\left(s',s\right) + b_{i+1}(s)\right)\right) \\ &= \underset{(s',s)/d_i \equiv \widehat{j}}{max^*}\left(a_i\left(s'\right) + c_i\left(s',s\right) + b_{i+1}(s)\right) \end{aligned}$$

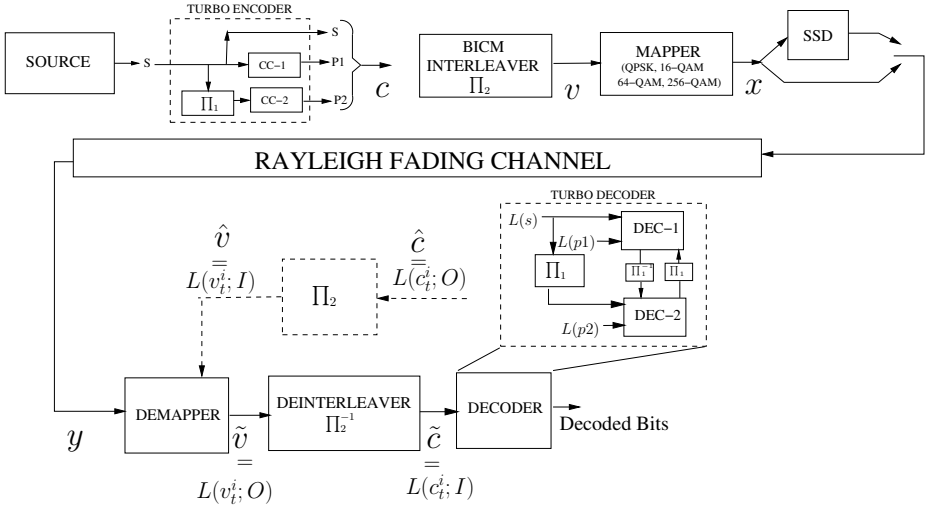


Figure 2.2 — System diagram with turbo encoder, BICM interleaver, mapper (optional SSD) Rayleigh fading channel, demapper, deinterleaver, and turbo decoder

and by distributivity :

$$\underset{(s',s)/d_i \equiv j}{max^*} \left(a_i\left(s'\right) + c_i\left(s', s\right) + b_{i+1}\left(s\right)\right) = \tag{2.17}$$
$$L_i^a(j) + L_i^{sys}(j) + \underset{(s',s)/d_i \equiv j}{max^*} \left(a_i\left(s'\right) + c_i^{ex}\left(s', s\right) + b_{i+1}\left(s\right)\right)$$

In the context of this simplification the metrics can be written in the following way:

$$L_i(j) = L_i^a(j) + L_i^{sys}(j) + L_i^{ex}(j) - L_i^a\left(\widehat{j}\right) - L_i^{sys}\left(\widehat{j}\right) \tag{2.18}$$

This simplification also known as log-MAP introduces a very low loss of performance (0.05dB) as compared to MAP algorithm. Although sub-optimal max-log-MAP algorithms provides a loss of 0.1dB for double binary code [30] yet it provides many advantages. Firstly it eliminates the logarithmic term from the expression (2.9) which on one hand reduces the implementation complexity (in practice this part is saved in Look Up Tables) and on the other hand reduces the critical path and hence increases the operational frequency as compared to max^* operator. Moreover, for max-log-MAP algorithm the knowledge of σ^2 is not required.

2.2 SISO Demapping Algorithm

In this section the basic system, considered in previous subsection, has been further extended to include BICM interleaver as shown in Fig.2.2. The other enhancements to the system are higher modulation orders (up to 256-QAM) and possibility of applying SSD (constellation rotation and time delay between I and Q component of a symbol). The channel is Rayleigh fading which includes fading factor ρ along with AWGN w with a variance of σ^2. Hence the signal y received at time instant t is given by:

$$y_t = \rho_t . x_t + w \tag{2.19}$$

On the receiver side there is a demapper (also called as demodulator) which produces the probabilities on transmit bit sequence $\mathbf{v}^i, i = \{0, ..., m-1\}$. At time t, the probability of error on bit v_t^i noted

$P(v_t^i = b|\mathbf{y}; O)$ is expressed as follows [9]:

$$P(v_t^i = b|\mathbf{y}; O) = \sum_{x_t \in \mathcal{X}_b^i} P(x_t|y_t) = \sum_{x_t \in \mathcal{X}_b^i} P(y_t|x_t).P(x_t) \tag{2.20}$$

where $b = \{0, 1\}$, $\mathcal{X}_b^i$ is the symbol set of constellation where i^{th} bit of x_t has value b, and $P(x_t)$ designates the *a priori* probability of x_t. In the presence of equi-probable source the $P(x_t) = 1$. After BICM deinterleaving the $P(v_t^i = b|\mathbf{y}; O)$, the resultant $P(c_t^i = b|\mathbf{y}; I)$ is the input to the SISO decoder. In case of BICM-ID the SISO decoder outputs the extrinsic information $\hat{c} = P(c_t^i = b|\mathbf{y}; O)$ which after interleaving becomes $\hat{v} = P(v_t^i = b|\mathbf{y}; I)$ and serves as *a priori* information on bits v_t^i. Using channel values y, fading coefficient ρ and $P(v_t^i = b|\mathbf{y}; I)$ the two parts of (2.20) can be computed as follows [10]:

$$P(y_t|x_t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2}} \tag{2.21}$$

$$P(x_t) = \prod_{\substack{l=0 \\ l \neq i}}^{m-1} P(v_t^l; I) \tag{2.22}$$

where d is the delay between the transmission of I and Q parts of a modulated symbol x while implementing SSD.

These expressions are considerably complex for hardware implementation, hence certain simplifications are available which are detailed in following sub sections. To better explain these simplifications for practical system implementations first a new term called Log Likelihood Ratio (LLR) is explained below.

2.2.1 Log Likelihood Ratio

LLR is the ratio of the probability of a bit to be one to the probability of that bit to be zero in logarithmic domain and is define as:

$$L(v_t^i; O) = ln\frac{P(v_t^i = 1|\mathbf{y}; O)}{P(v_t^i = 0|\mathbf{y}; O)} = ln(P(v_t^i = 1|\mathbf{y}; O)) - ln(P(v_t^i = 0|\mathbf{y}; O)) \tag{2.23}$$

Consider the system of Fig.2.2 where the demapper outputs the LLRs which, after deinterleaving, become the input to the decoder. The decoder which is also working in the logarithmic domain, takes $L(c_t^i; I)$ as input and outputs extrinsic information, $L(c_t^i; O)$, which after interleaving, $L(v_t^i; I)$, serve as *a priori* information to the demapper.

2.2.2 Simplification of $P(x_t)$

To compute the expression of (2.22) from the input $L(v_t^i; I)$, first we need to change these LLRs back into probability which can be done using following expression:

$$P(v_t^i = 1; I) = \frac{e^{L(v_t^i;I)}}{1 + e^{L(v_t^i;I)}} \tag{2.24}$$

$$P(v_t^i = 0; I) = \frac{1}{1 + e^{L(v_t^i;I)}} \tag{2.25}$$

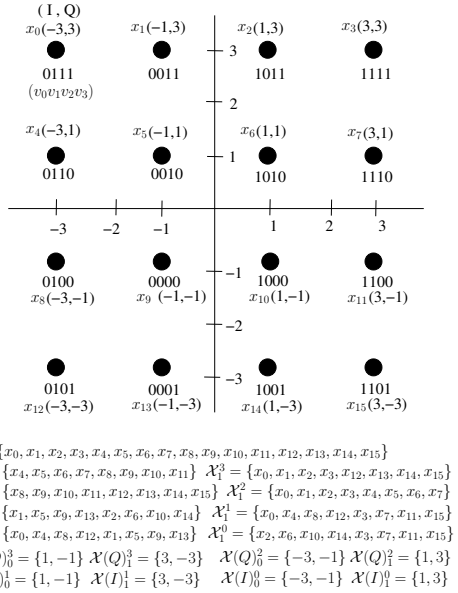


Figure 2.3 — Example of Gray mapped 16-QAM constellation

To explain the simplification, consider a received symbol y of 16-QAM constellation and the symbol x_t with binary mapping 0101 $(v_t^0, v_t^1, v_t^2, v_t^3)$ from symbol set $\mathcal{X}_0^0$ as shown in Fig.2.3. The computation of $P(x_t)$, using (2.22), is given by:

$$P(x_t = 0101; x_t \in \mathcal{X}_0^0) = P(v_t^1 = 1; I).P(v_t^2 = 0; I).P(v_t^3 = 1; I)$$

Normalizing each probability with its value equal to 0, the expression becomes:

$$\begin{aligned} P(x_t = 0101 \in \mathcal{X}_0^0) &\approx \frac{P(v_t^1 = 1; I).P(v_t^2 = 0; I).P(v_t^3 = 1; I)}{P(v_t^1 = 0; I).P(v_t^2 = 0; I).P(v_t^3 = 0; I)} && (2.26)\\ &\approx e^{L(v_t^1;I)}.e^{L(v_t^3;I)} && (2.27)\\ &\approx e^{L(v_t^1;I)+L(v_t^3;I)} && (2.28) \end{aligned}$$

The expression can be generalized as:

$$P(x_t; x_t \in \mathcal{X}_b^i) \approx e^{\sum_{l=0,\, l\neq i \text{ and } v_t^l=1}^{m-1} L(v_t^l;I)} \qquad (2.29)$$

2.2.3 The max-log Approximation

Using (2.21) and (2.29), expression of (2.20) becomes:

$$P(v_t^i = b|\mathbf{y};O) \approx \sum_{x_t \in \mathcal{X}_b^i} \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2}\right)} .e^{\left(\sum\limits_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right)} \quad (2.30)$$

$$\approx \sum_{x_t \in \mathcal{X}_b^i} \frac{1}{\sigma\sqrt{2\pi}} e^{\left(-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} + \sum\limits_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right)}$$

Taking natural log of (2.30) to be used in (2.23), the expression becomes:

$$ln\left(P(v_t^i = b|\mathbf{y};O)\right) \approx K + ln\left(\sum_{x_t \in \mathcal{X}_b^i} e^{\left(-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} + \sum\limits_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right)}\right) \quad (2.31)$$

Applying the log-max approximation as given in (2.9), expression in (2.31) becomes:

$$ln\left(P(v_t^i = b|\mathbf{y};O)\right) \approx K + \max_{x_t \in \mathcal{X}_b^i}\left(-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} + \sum_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right) \quad (2.32)$$

Hence the LLR expression becomes

$$L(v_t^i;O) \approx \max_{x_t \in \mathcal{X}_1^i}\left(-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} + \sum_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right) \quad (2.33)$$

$$- \max_{x_t \in \mathcal{X}_0^i}\left(-\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} + \sum_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right)$$

which can be rewritten as:

$$L(v_t^i;O) \approx \min_{x_t \in \mathcal{X}_0^i}\left(\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} - \sum_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right) \quad (2.34)$$

$$- \min_{x_t \in \mathcal{X}_1^i}\left(\frac{|y_{t-d}^I - \rho_{t-d}^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} - \sum_{\substack{l=0 \\ l\neq i \text{ and } v_t^l=1}}^{m-1} L(v_t^l;I)\right)$$

Hence by applying the max-log approximation the sum of exponentials in (2.20) reduces to minimum operations to compute the LLRs for the decoder.

2.2.4 Simplification For Gray Mapped Constellation

In the absence of *a priori* information and no SSD the expression of LLR in (2.34)is :

$$L(v_t^i;O) \approx \min_{x_t \in \mathcal{X}_0^i} \left(\frac{|y_t^I - \rho_t^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} \right) - \min_{x_t \in \mathcal{X}_1^i} \left(\frac{|y_t^I - \rho_t^I.x^I|^2 + |y_t^Q - \rho_t^Q.x_Q|^2}{2\sigma^2} \right) \tag{2.35}$$

In case of a constellation which is Gray mapped and m is even, the I and Q components are independent of each other. A simplification has been addressed in [**?**] which transforms the LLR computation expression into:

$$L(v_t^i;O) \approx \frac{1}{2\sigma^2} \left[\min_{x_t^I \in \mathcal{X}(I)_0^i} \left(|y_t^I - \rho_t.x_t^I|^2 \right) - \min_{x_t^I \in \mathcal{X}(I)_1^i} \left(|y_t^I - \rho_t.x_t^I|^2 \right) \right] \text{ for } i = 0 \dots \frac{m}{2} - 1 \tag{2.36}$$

and

$$L(v_t^j;O) \approx \frac{1}{2\sigma^2} \left[\min_{x_t^{(Q)} \in \mathcal{X}(Q)_0^j} \left(|y_t^Q - \rho_t.x_t^Q|^2 \right) - \min_{x_t^Q \in \mathcal{X}(Q)_1^j} \left(|y_t^Q - \rho_t.x_t^Q|^2 \right) \right] \text{ for } j = \frac{m}{2} \dots m - 1 \tag{2.37}$$

The $\mathcal{X}(I)_b^i$ and $\mathcal{X}(Q)_b^j$ are the point sets on I and Q-axis where i^{th} and j^{th} bits of symbol x_t has value equal to b.

2.3 SISO Equalization Algorithm

Let us now consider the system model of Fig. 2.4 which extend the system model of Fig. 2.2 by including by including MIMO STC in transmitter to achieve the spatial multiplexing and/or time diversity. The input vector $\mathbf{y}$ received at the input of the receiver is given by the following expression:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{w} \tag{2.38}$$

where $\mathbf{H}$ is the MIMO channel matrix and $\mathbf{w}$ is AWGN noise vector. On the receiver side a MIMO equalizer has been added which removes the co-antenna interference and provides the estimated symbol vector $\tilde{\mathbf{x}}$, corresponding equivalent bias vector (fading coefficient) and equivalent noise variance to the demapper. From the demapper, the forward path works as described in previous subsection. Regarding feedback path, the turbo decoder having the capability of producing both a *a posteriori* and/or *extrinsic* information, provides *a posteriori* information to a soft mapper which provides the *a priori* information to the equalizer as decoded symbol vector $\hat{\mathbf{x}}$. Inside the equalizer the two informations, received channel symbol vectors and decoded symbol vectors, are used as described in following subsections.

2.3.1 MMSE-IC LE Algorithm

The equalizer considers that a symbol of the vector $\mathbf{x}$ is distorted by the $N_t - 1$ other symbols of the vector and the channel noise. Hence the role of the equalizer is to battle both multi antenna

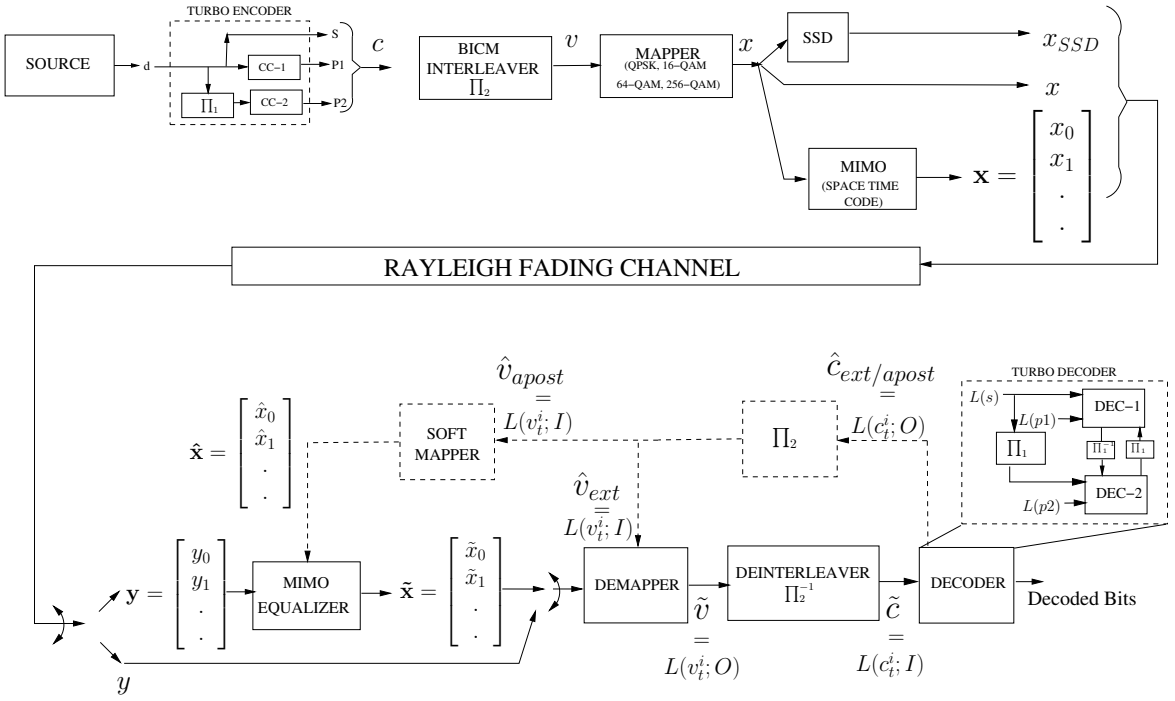


Figure 2.4 — System diagram with turbo encoder, BICM interleaver, mapper (optional SSD), STC, Rayleigh fading channel, MIMO equalizer, demapper, deinterleaver, turbo decoder and soft mapper

interference and channel noise. The right side of expression (2.38) can be divided into desired symbol x_j and multi-antenna interference:

$$\mathbf{y} = \underbrace{\mathbf{h}_j.x_j}_{\text{useful signal}} + \underbrace{\sum_{i \neq j} \mathbf{h}_i.x_i}_{\text{interference}} + \underbrace{\mathbf{w}}_{\text{noise}}, \quad j \in \{0, N_t - 1\} \tag{2.39}$$

where $\mathbf{h}_j$ is the j^{th} column of $\mathbf{H}$ matrix. One of the low complexity techniques to achieve the equalization function is the use of filter based symbol equalization [31]. An estimation of the symbol x_j can be carried out through a linear filter which minimizes the mean square error (MMSE) between the transmitted symbol x_j and the output of the equalizer $\tilde{x}_j$. The estimated symbol can be presented in the following form, using the optimal Wiener filter $\mathbf{a}_j^H = \lambda_j.\mathbf{p}_j^H$ [32][33][34]:

$$\begin{aligned} \tilde{x}_j &= \mathbf{a}_j^H(\mathbf{y} - \sum_{i \neq j} \mathbf{h}_i.\hat{x}_i) \\ &= \lambda_j.\mathbf{p}_j^H(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}} + \mathbf{h}_j\hat{x}_j) \\ &= \lambda_j[\mathbf{p}_j^H(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}) + \beta_j\hat{x}_j] \\ &= \lambda_j.\mathbf{p}_j^H(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}}) + g_j\hat{x}_j \end{aligned} \tag{2.40}$$

with :

$$\begin{aligned}
\beta_j &= \mathbf{p}_j^H \mathbf{h}_j \\
\mathbf{p}_j &= (\mathbf{H}\mathcal{U}\mathbf{H}^H + \sigma_w^2.\mathbf{I})^{-1}\mathbf{h}_j \\
\mathcal{U} &= \text{diag}(\hat{u}_0^2, \ldots, \hat{u}_{N_t-1}^2) \\
\hat{u}_j^2 &= \mathbb{E}\left\{ |x_j - \hat{x}_j|^2 \, |\mathcal{L}_a \right\} \\
g_j &= \lambda_j.\beta_j \\
\lambda_j &= \frac{\sigma_x^2}{1 + (\sigma_x^2 - \hat{u}_j^2)\beta_j}
\end{aligned} \tag{2.41}$$

where $j \in \{0, N_t - 1\}$.

- λ_j, β_j et g_j represent the MMSE equalization coefficients,
- $\mathbf{p}_j$ refers to MMSE detection vector,
- σ_x^2 is the variance of the constellation of transmitted symbols,
- the coefficient $\hat{u}_j^2$ represents a reliability information reflecting the residual error on the decoded symbol $\hat{x}_j$ as compared to the symbols of the constellation.

The diagonal matrix $\mathcal{U} = \text{diag}(\hat{u}_0^2, \ldots, \hat{u}_{N_t-1}^2)$ is made up of $\hat{u}_j^2$ and is involved in the computation of $\mathbf{p}_j$ vector. During the first iteration of turbo equalization process, no *a priori* information of the transmitted symbols are available ($\hat{x}_j = 0$) and the symbols are equiprobable. Therefore each term $\hat{u}_j^2$ becomes equal to σ_x^2, and $\tilde{x}_j$ can be written as:

$$\tilde{x}_j = \sigma_x^2 \mathbf{h}_j^H (\sigma_x^2 \mathbf{H}.\mathbf{H}^H + \sigma_w^2.\mathbf{I})^{-1}\mathbf{y} \tag{2.42}$$

From the second iteration, the *a priori* information provided by channel decoder about transmitted symbols, gradually improves over the iterations and approaches to asymptotic performance. The asymptotic performance is achieved when the estimated data becomes equal to the transmitted data ($\hat{x}_j = x_j$), which is also called the 'Geni' case. By replacing $\hat{x}_j$ by x_j in equation (2.40), the detected symbol can be written:

$$\tilde{x}_j = \frac{\sigma_x^2 \mathbf{h}_j^H}{\sigma_x^2 \mathbf{h}_j^H \mathbf{h}_j + \sigma_w^2}(\mathbf{h}_j \hat{x}_j + \mathbf{w})$$

The *a priori* information $\hat{x}_j$ is calculated from the LLRs produced by channel decoder. One can divide the (2.40) into two terms: the first one reflecting the effect of interference cancellation (IC) and the other related to the desired symbol by using bias or equivalent fading coefficient g_j. The computation of $\mathbf{p}_j = (\mathbf{H}\mathcal{U}\mathbf{H}^H + \sigma_w^2.\mathbf{I})^{-1}\mathbf{h}_j = \mathbf{E}^{-1}.\mathbf{h}_j = \mathcal{F}.\mathbf{h}_j$ leads to matrix inversion [32][33]. The matrix $\mathcal{F}$ of size $N_r \times N_r$ is calculated for each space time block resulting in a significant computational complexity. It is however possible to reduce the complexity of the algorithm of MMSE-IC detection by simplifying the calculation of matrix inversion or limiting its use. To do this, approximate versions of MMSE-IC algorithm can be used: MMSE-IC$_1$, MMSE-IC$_2$.

2.3.1.1 MMSE-IC$_1$

A first approximation of the MMSE-IC algorithm [35][36] is to simplify the inverse matrix by replacing the terms $\hat{u}_j^2$ by its average value $\hat{u}^2$. It has been shown in [33] that :

$$\begin{aligned}\hat{u}^2 = \mathbb{E}\left\{\hat{u}_j^2\right\} &= \mathbb{E}\left\{\mathbb{E}\left\{|x_j - \hat{x}_j|^2 \,|\mathcal{L}_a\right\}\right\} \\ &= \mathbb{E}\left\{\mathbb{E}\left\{|x_j|^2 \,|\mathcal{L}_a\right\}\right\} - \mathbb{E}\left\{|\hat{x}_j|^2\right\} \\ &= \mathbb{E}\left\{|x_j|^2\right\} - \mathbb{E}\left\{|\hat{x}_j|^2\right\} \\ &= \sigma_x^2 - \sigma_{\hat{x}}^2\end{aligned} \tag{2.43}$$

where $\sigma_{\hat{x}}^2$ represents the variance of the decoded symbols. By applying this approximation, the matrix $\mathcal{F}$ becomes a hermitian matrix.

$$\mathcal{F} = E^{-1} = ((\sigma_x^2 - \sigma_{\hat{x}}^2)\mathbf{H}.\mathbf{H}^H + \sigma_w^2.\mathbf{I})^{-1} \tag{2.44}$$

and

$$\mathbf{p}_j = \mathcal{F}\mathbf{h}_j \tag{2.45}$$

Likewise, in (2.41), by replacing the term $\hat{u}_j^2$ by its average value $\hat{u}^2$, the coefficients λ_j becomes :

$$\lambda_j = \frac{\sigma_x^2}{1 + \sigma_{\hat{x}}^2 \beta_j} \tag{2.46}$$

This simplified version requires less computation than the original algorithm since the matrix $\mathcal{F}$ does not depend on the terms $\hat{u}_j^2$, but on their average $\hat{u}^2$. Moreover, taking into account the properties of the matrix $\mathcal{F}$ we can greatly simplify the matrix inversion, e.g. the determinant of a Hermitian matrix is real. The division by the determinant is used explicitly or implicitly by the matrix inversion method chosen. Hence, having a real determinant rather than the complex one is interesting to reduce the complexity of the matrix inversion implementation.

2.3.1.2 MMSE-IC$_2$

By considering one of the *a priori* information perfect ($\sigma_{\hat{x}}^2 = \sigma_x^2$) from second iteration, the matrix inversion is necessary only in the first iteration (see equation 2.42). For the next iterations the $\tilde{x}_j$ are given by :

$$\tilde{x}_j = \frac{\sigma_x^2 \mathbf{h}_j^H}{\sigma_x^2 \mathbf{h}_j^H \mathbf{h}_j + \sigma_w^2}(\mathbf{y} - \mathbf{H}\hat{\mathbf{x}} + \mathbf{h}_j \hat{x}_j) \tag{2.47}$$

This method [37] returns to a conventional MMSE detection information without *a priori* in the first iteration and then use the same linear filtering (matched filtering type) for all subsequent iterations. It has a lower complexity as compared to MMSE-IC$_1$. However, this has an impact on the performance, as out of two *a priori* informations ($\sigma_{\hat{x}}^2$ and $\hat{x}$), $\sigma_{\hat{x}}^2$ has been forced to its maximum value. Note that the linear filter used to detect IC-MMSE$_2$ is equivalent to zero forcing detection with *a priori* information (ZF-IC: *Zero Forcing - Interference Canceler*) or that of the matched filter (MF-IC: *Matched Filter - Interference Canceler*) [38].

2.3.2 Soft Demapping

To demap an estimated symbol $\tilde{x}$, the expression (2.35) is used where symbol y is replaced with $\tilde{x}$, ρ will be replaced with g and σ^2 is replaced with $g(1-g)\sigma_x^2$. In case of Gray mapped constellation, expressions (2.36) and (2.37) will provide a simplified expression with similar replacements of y, ρ and σ^2 with $\tilde{x}$, g and $g(1-g)\sigma_x^2$ respectively.

2.3.3 Soft Mapping

The LLR to symbol conversion is carried out with the help of LLRs ($\tilde{v}_{apost} = L(v_t^i; I)$) coming out of the BICM interleaver ($\prod_2$) in feedback path. The estimation of $\hat{x}$ of transmitted symbol x is given by :

$$\begin{aligned} \hat{x} &= \mathbb{E}\{x|\mathcal{L}_a\} \\ &= \sum_{s\in\mathcal{X}} s\mathrm{P}\{x = s|\mathcal{L}_a\} \end{aligned} \tag{2.48}$$

where $\mathcal{L}_a$ is the subset of LLRs ($\hat{v}_{apost} = L(v_t^i; I)$) corresponding to bits constituting the transmitted symbol x which is part of constellation $\mathcal{X}$ of size 2^m. The term $\mathrm{P}\{x = s|\mathcal{L}_a\}$ means the *a priori* probability of symbol s. By assuming the transmitted bits statistically independent this probability becomes:

$$\mathrm{P}\{x = s|\mathcal{L}_a\} = \prod_{i=0}^{m-1} \mathrm{P}\left\{v^i = b\right\} \tag{2.49}$$

where v^i is the i^{th} bit of symbol s having value $b = \{0, 1\}$ according to constellation mapping used and the term $\mathrm{P}\left\{v^i = b\right\}$ can be computed using (2.24) and (2.25).

2.4 Parallelism in Turbo Receiver

Iterative/turbo processing in a wireless base-band receiver ensures promising error rate performance at the cost of high processing time for data retrieval. Exploration of parallelism in iterative processes, while maintaining the error rate performance, contributes towards fulfilling the high transmission rate requirements of emerging wireless communication standards. This section is dedicated to illustrate the available parallelism in the three previously described turbo processes (turbo decoding, turbo demodulation and turbo equalization). A comprehensive study on parallelism levels for turbo decoding is already conducted in [39] and will be briefly summarized in next subsection. However, regarding turbo demodulation and turbo equalization, the study on parallelism is conducted for the first time and presented here. Parallelism classification is presented in this section, while parallel system modeling and simulation results are detailed in section 2.5.

2.4.1 Parallelism in Turbo Decoding

In turbo decoding with the max-log-MAP algorithm executing in the decoder, the parallelism can be classified at three levels: (1) Metric level, (2) SISO decoder level, and (3) Turbo-decoding level. The first (lowest) parallelism level concerns the elementary computations for LLR generation inside a SISO decoder. Parallelism between these SISO components, inside a turbo decoding process, belongs to the second parallelism level. The third (highest) parallelism level duplicates the whole turbo decoder hardware itself.

2.4.1.1 Metric Level Parallelism

The level of parallelism of BCJR metrics computation addresses the parallelism available in the calculations of all the metrics required to decode each received symbol within a BCJR SISO decoder. This level of parallelism exploits both the parallelism of inherent structure of the trellis [40][41], and the parallel calculations of BCJR [40][42][41].

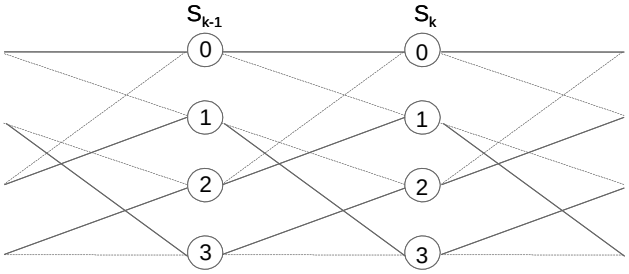


Figure 2.5 — Example of trellis

Parallelism of Trellis Transition: In a decoder, the first parallelism available in the max-log-MAP algorithm is computation of the metrics associated to each transition of a trellis regarding γ, α, β, and the extrinsic information. Trellis-transition parallelism can easily be extracted from the trellis structure as the same operations are repeated for all transitions. The first metric (γ) calculation is completely parallelizable with a degree of parallelism naturally bounded by the number of transitions in the trellis. But in practice, the degree of parallelism associated with calculating the branch metric is bounded by the number of possible binary combinations of input and parity bits. Thus, several transitions may have the same probability in a trellis. The other metrics α, β, and extrinsic computation can be parallelized with a bound of total number of transitions in a trellis. Furthermore this parallelism implies low area overhead as only the computational units have to be duplicated. In particular, no additional memories are required since all the parallelized operations are executed on the same trellis section, and in consequence on the same data.

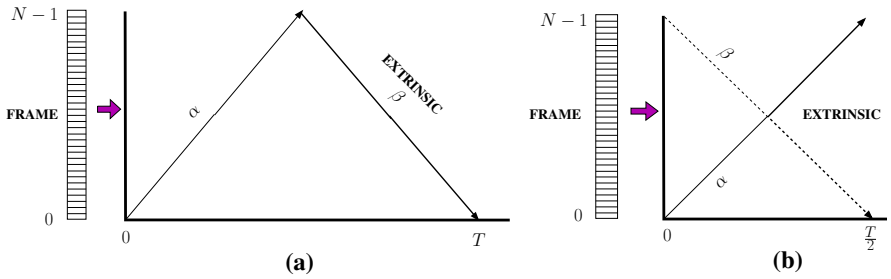


Figure 2.6 — (a) Forward backward scheme (b) Butterfly scheme

Parallelism of BCJR Calculation: A second metric parallelism can be orthogonally extracted from the BCJR algorithm through a parallel execution of the three BCJR computations (α, β, and extrinsic computation). Parallel execution of backward recursion and extrinsic computations was proposed with the original Forward-Backward scheme, depicted in Fig.2.6(a). So, in this scheme, we can notice that BCJR computation parallelism degree is equal to one in the forward part and two in the backward part.

To increase this parallelism degree, several schemes are proposed [42]. Fig.2.6(b) shows the butterfly scheme which doubles the parallelism degree of the original scheme through the parallelism

between the forward and backward recursion computations. This is performed without any memory increase and only BCJR computation resources have to be duplicated. Thus, metric computation parallelism is area efficient but still limited in parallelism degree.

2.4.1.2 SISO Decoder Level Parallelism

The second level of parallelism concerns the SISO decoder level. It consists of the use of multiple SISO decoders, each executing the BCJR algorithm and processing a sub-block of the same frame in one of the two interleaving orders. At this level, parallelism can be applied either on sub-blocks and/or on component decoders.

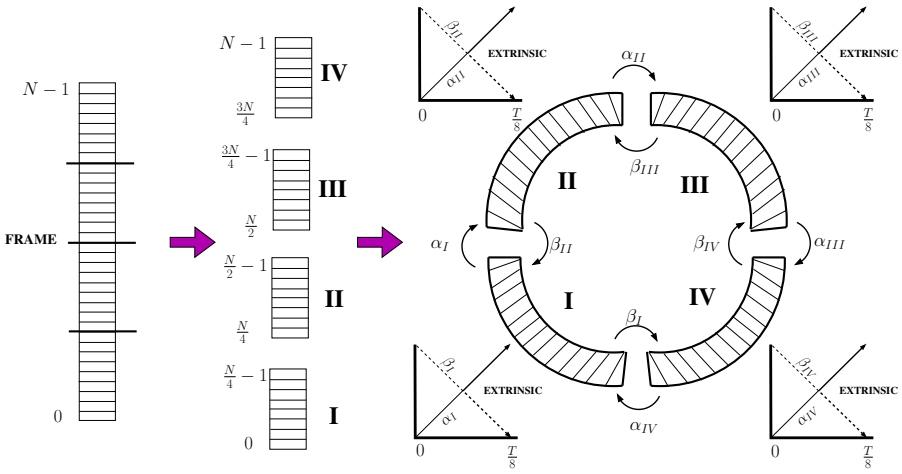


Figure 2.7 — Sub-block parallelism with message passing for metric initialization

Frame Sub-blocking: In sub-block parallelism, each frame is divided into M sub-blocks and then each sub-block is processed on a BCJR-SISO decoder using adequate initializations as shown in Fig.2.7. Besides duplication of BCJR-SISO decoders, this parallelism imposes two other constraints. On the one hand, interleaving has to be parallelized in order to scale proportionally the communication bandwidth. Due to the scramble property of interleaving, this parallelism can induce communication conflicts except for interleavers of emerging standards that are conflict-free for certain parallelism degrees. These conflicts force the communication structure to implement conflict management mechanisms and imply a long and variable communication time. This issue is generally addressed by minimizing interleaving delay with specific communication networks [43]. On the other hand, BCJR-SISO decoders have to be initialized adequately either by acquisition or by message passing.

Acquisition method has two implications on implementation. First of all extra memory is required to store the overlapping windows when frame sub-blocking is used and secondly extra time will be required for performing acquisition. Other method, the message passing, which initializes a sub-block with recursion metrics computed during the previous iteration in the neighboring sub-blocks, needs not to store the recursion metric and time overhead is negligible. In [39] a detailed analysis of the parallelism efficiency of these two methods is presented which gives favor to the use of message passing technique.

Shuffled Turbo Decoding : The basic idea of shuffled decoding technique [44] is to execute all component decoders in parallel and to exchange extrinsic information as soon as it is created, so that component decoders use more reliable a priori information. Thus the shuffled decoding technique performs decoding (computation time) and interleaving (communication time) fully concurrently while

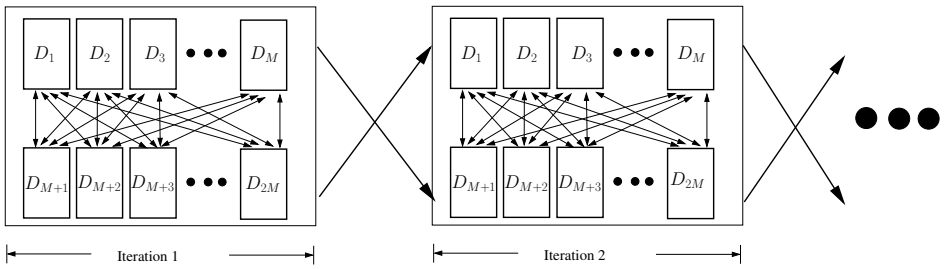


Figure 2.8 — Shuffled turbo decoding

serial decoding implies waiting for the update of all extrinsic information before starting the next half iteration (Fig.2.8). Thus, by doubling the number of BCJR SISO decoders, component-decoder parallelism halves the iteration period in comparison with originally proposed serial turbo decoding.

Nevertheless,to preserve error-rate performance with shuffled turbo decoding, an overhead of iteration between 5 and 50 percent is required depending on the BCJR computation scheme, on the degree of sub-block parallelism, on propagation time, and on interleaving rules [39].

2.4.1.3 Parallelism of Turbo Decoder

The highest level of parallelism simply duplicates whole turbo decoders to process iterations and/or frames in parallel. Iteration parallelism occurs in a pipelined fashion with a maximum pipeline depth equal to the iteration number, whereas frame parallelism presents no limitation in parallelism degree. Nevertheless, turbo-decoder level parallelism is too area-expensive (all memories and computation resources are duplicated) and presents no gain in frame decoding latency.

2.4.2 Parallelism in Turbo Demodulation

In a turbo demodulation system, turbo decoding parallelism presented above can be directly inherited in the decoding part. In demodulator part, the parallelism can be categorized into same three levels as defined in turbo decoding which are discussed below.

2.4.2.1 Metric Level Parallelism

Generation of LLR using (2.34) needs Euclidean distances between received symbol y and constellation symbols x, the sum of *a priori* information and minimum operations. The metric level parallelism concerns the concurrent computations of all of these operations. In a constellation with m bits per symbol, one needs 2^m distance and $a\ priori$ addition operations which are then fed to $2m$ minimum operations each having 2^{m-1} inputs. This illustrates the maximum parallelism degree at this level, which varies significantly with m, and thus, poses a real limitation while targeting a flexible, yet efficient architecture.

If fact, having minimum operations over more than two input values will imply a cascaded hardware min operators which can significantly impact the maximum achievable clock frequency. Thus using 2-input minimum operations will improve both performance in terms of clock speed and flexibility efficiency for different constellation sizes. In order to support up to 256-QAM constellation, the parallelism degree regarding the required minimum operations (2.34) is limited to 16. In this context, if two distances between received symbol y and two x symbols with complementary binary

mapping is computed, all the minimum operations related to each LLR of received symbol y can be performed concurrently. Hence, two computational units constitute the most efficient parallelism degree for distance calculators and *a priori* adders at this level.

2.4.2.2 Demapper Component Level Parallelism

Same as in turbo decoding, there are two categories at this level of parallelism: sub-block parallelism and shuffled turbo demodulation.

Frame Sub-blocking: At this level, the feature which can be exploited for parallelism in the demapper is the independence of a symbol from other symbols in a frame received from a memoryless channel. Hence, in demapper there are no issue of sub-block initialization at this level of parallelism compared to turbo decoding. Therefore, to achieve a linear increase in throughput by the addition of multiple distance calculators, which is limited to a parallelism degree of two at the first parallelism level, the best choice is to demap symbols from multiple sub-blocks in parallel. This parallelism level also imposes the requirement of an adequate communication network to resolve the conflicts caused by the interleaving property of BICM.

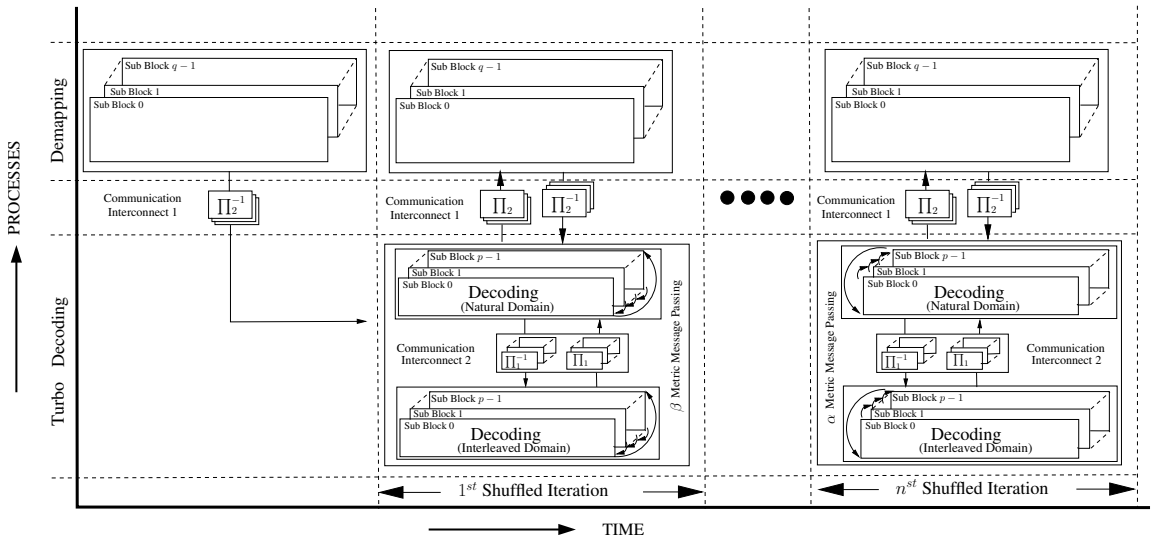


Figure 2.9 — Proposed execution of parallel turbo demodulation

Shuffled Turbo Demodulation: This type of parallelism is inherited from the concept of shuffled turbo decoding to execute both the decoding and demodulation tasks concurrently. The proposed scheduling of shuffled turbo demodulation process is shown in Fig.2.9. Once the demapper components receive the input data, demapping is performed for the first time without a priori information to fill the memories of the decoder components. After this, both processes run in a shuffled scheme and exchange information with other SISO components.

2.4.2.3 Turbo Demodulation Level Parallelism

The highest level of parallelism duplicates the whole turbo demodulator to process iterations and/or frames in parallel.

2.4.3 Parallelism in Turbo Equalization

The proposed three level classification of parallelism techniques is extended to turbo equalization and detailed below.

2.4.3.1 Symbol Estimation Level Parallelism

A closer look at the expression required in MMSE-IC algorithm (2.40 to 2.47) reveals the serial nature of the implied elementary computations. Due to this fact, the only parallelism possibility in MMSE-IC algorithm at this level is temporal parallelism which can be achieved through pipelining technique. Hence, at top level one can temporally parallelize equalization coefficient computations and symbol estimation by pipelining them. Next step is to spread each of coefficient computation and symbol estimation process on different pipeline stages depending upon the critical path requirements. Once pipeline stages are decided one can fully parallelize each stage. The idea can be explained through an example of HH^H computation in (2.44) for an H matrix having 4 rows and 4 columns. In a 4×4 complex numbered matrix multiplication, to get one element of the resultant matrix one need to multiply 4 elements of a row of first matrix with 4 elements of a column of second matrix and then add the results of these four multiplications through two serial complex additions. Hence, computation of one element of HH^H needs 4 complex multipliers and three complex adders with a critical path consisting of one complex multiplication and 2 serial complex additions. The maximum parallelism degree for the 4×4 configuration, at this parallelism level, corresponds to the concurrent computation of the 16 elements of HH^H which requires 64 complex multipliers and 48 complex adders. Similar parallelism degrees can be achieved at this level for other pipeline stages of equalization coefficient computations and symbol estimation.

2.4.3.2 Equalizer Component Level Parallelism

Parallelism techniques at this level can be classified in two categories: sub-block parallelism and shuffled turbo equalization.

Frame Sub-blocking: At this level, the feature which can be exploited for parallelism in the equalizer is the independence of a symbol vector from other vectors in a frame received from a memory-less channel. Hence, a linear increase in throughput can be achieved by the addition of more equalizer components to process different sub-blocks concurrently. In consequence, multiple demapper and soft mapper components will be required to balance the throughput of the multiple equalizers.

Shuffled Turbo Equalization: The proposed scheduling of shuffled turbo equalization with sub-block parallelism is shown in Fig.2.10. Once the equalizer components receive symbol vector sub-blocks, they perform symbol estimation in the absence of *a priori* information. The associated demapper components generate the LLRs from these estimated symbols in a pipelined fashion, which are deinterleaved before filling the input data memories of the decoder. After filling the decoder memories, all components of the parallel turbo equalizer work concurrently. Soft mappers, equalizers and demappers work in pipeline fashion to generate LLRs for the decoder while, on the other side, decoder components generate LLRs for the equalization side. As soon as LLRs are generated by demapper components and decoder components they are exchanged. Computation of $\sigma^2_{\hat{x}}$ is carried out during the soft mapping process and hence used in next shuffle iteration.

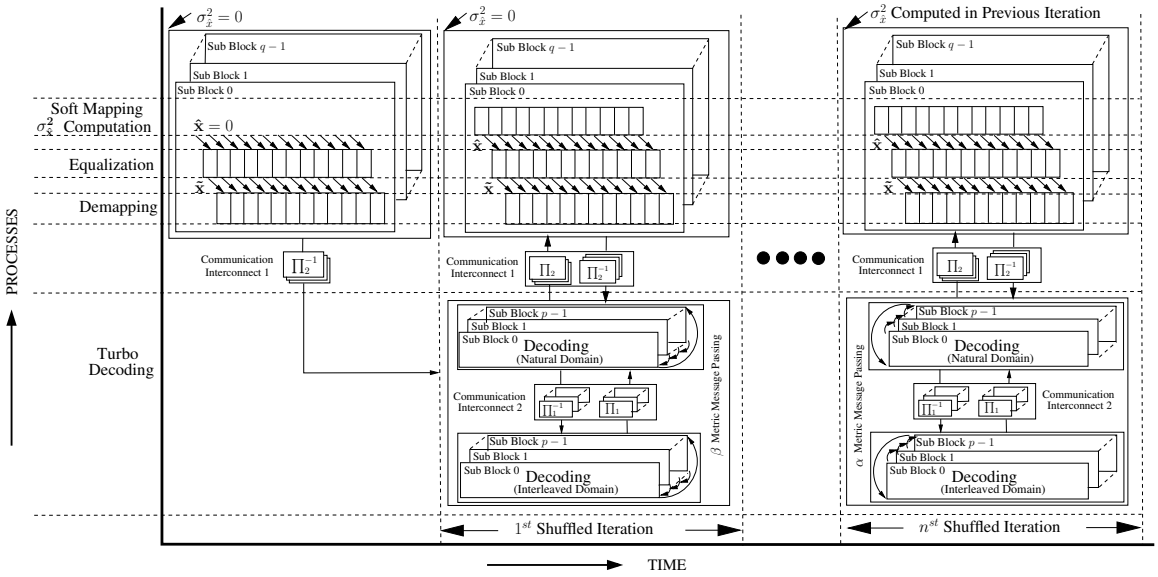


Figure 2.10 — Proposed execution of parallel turbo equalization

2.4.3.3 Turbo Equalization Level Parallelism

The highest level of parallelism duplicates the whole turbo equalization to process iterations and/or frames in parallel.

2.5 Parallel System Modeling and Simulation Results

The parallelism performance for convolutional turbo decoder has been comprehensively studied in [39]. In order to analyze the parallelism performance both for parallel turbo demodulation and parallel turbo equalization, corresponding software models are created which simulate the first and second level of parallelisms at LLR transaction level. Following subsections separately detail the software model and simulation results of parallel turbo demodulation and parallel turbo equalization.

2.5.1 Parallel Turbo Demodulation

Regarding the software model for parallel turbo demodulation, double binary turbo code of DVB-RCS standard, random BICM interleaver ($\prod_2$), mapper using QPSK, 16-QAM, 64-QAM and 256-QAM and SSD are modeled on the transmitter side. Random BICM interleaver is implemented in such a way that the systematic bits from coded bits are always mapped on the most protected bits of the modulated symbols. To implement SSD delay d is taken as one and variable rotation angles can be input to the model. The channel is implemented as Rayleigh fading.

2.5.1.1 Software Model for Parallel Turbo Demodulation

The modeled architecture of the parallel turbo demodulator is shown in Fig. 2.11. To better explain the functioning of the parallel turbo demodulator on the receiver side, consider an example of a frame having A source symbols where each symbol is made up of n bits encoded at a code rate r and then modulated with m bits per symbol. In this case, there will be $B = \frac{A \times n}{r \times m}$ modulated

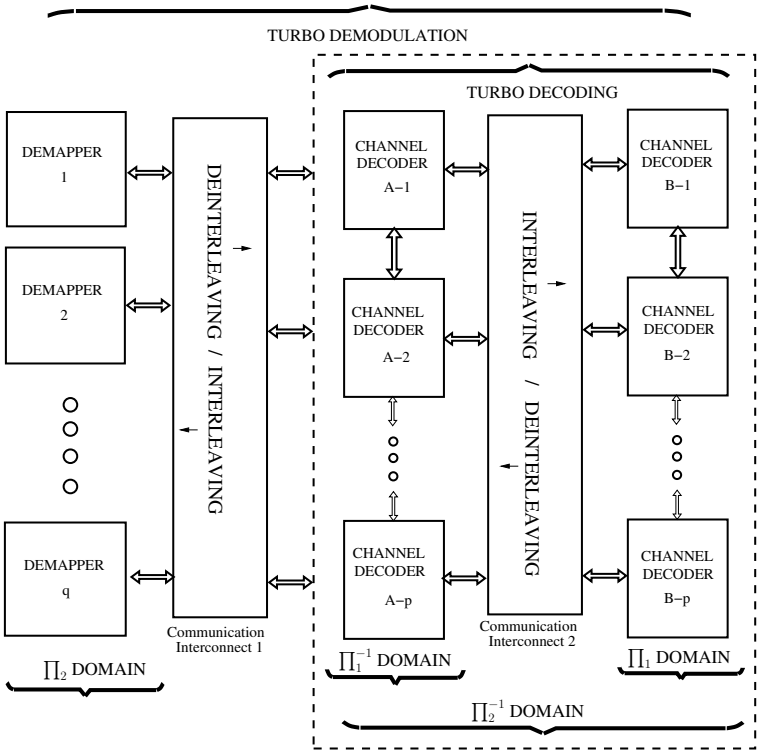


Figure 2.11 — Architecture of receiver's software model

symbols in the frame. In a shuffled turbo demodulation system, the ideal case is that both decoding and demapping tasks finish at the same time. Since the demapping and decoding tasks are different in nature, another parameter, the symbol processing throughput ratio of turbo decoder and demapper component "$tp_r = \frac{Decoder\ throughput}{Demapper\ throughput}$" is required. In this scenario the ratio of turbo decoder and demapper components (DD_r) required to finish the frame at the same time can be deduced as follows:

$$\text{Time for decoding = Time for demapping} \tag{2.50}$$

$$\frac{\frac{A}{No.\ of\ Decoders\ (p)}}{Decoder\ throughput} = \frac{\frac{B}{No.\ of\ Demappers\ (q)}}{Demapper\ throughput} \tag{2.51}$$

$$DD_r = \frac{No.\ of\ Decoders\ (p)}{No.\ of\ Demappers\ (q)} = \frac{A}{B \times tp_r} = \frac{r \times m}{n \times tp_r} \tag{2.52}$$

Using the above expression, for a dual binary turbo encoder with code rate $r = 0.5$, 256-QAM system and $tp_r = 1$ the parameter $DD_r = 2$, i.e, for one demapper component two decoder components on each side of the turbo decoder are required, whereas for 16-QAM this ratio is one.

Based on this idea, the number of demappers and decoders are given to the software model and sub-blocks of modulated and encoded symbols are assigned to these components respectively. For the first time multiple demappers work in parallel and provide the inputs to the decoder after BICM deinterleaving (using communication network). Later on all demapper and decoder components execute together. In case that tp_r is one, at each time unit, each demapper component demaps one modulated symbol. In the turbo decoder, if a decoder component is on the left side of the butterfly (Fig.2.7) each decoder computes α and β metrics of two symbols and on the right side it computes α, β and extrinsic information for two symbols per unit of time. Once the LLRs are ready, exchange of information through the interleaver/deinterleaver networks of the BICM and that of the turbo decoder takes place

which implements the second level of parallelism. To achieve a scenario where demapping and decoding tasks do not finish at the same time, either due to lesser components than required or due to mismatch in throughput, the faster side will process the whole frame in an iteration while the other will do the same job in multiple iterations.

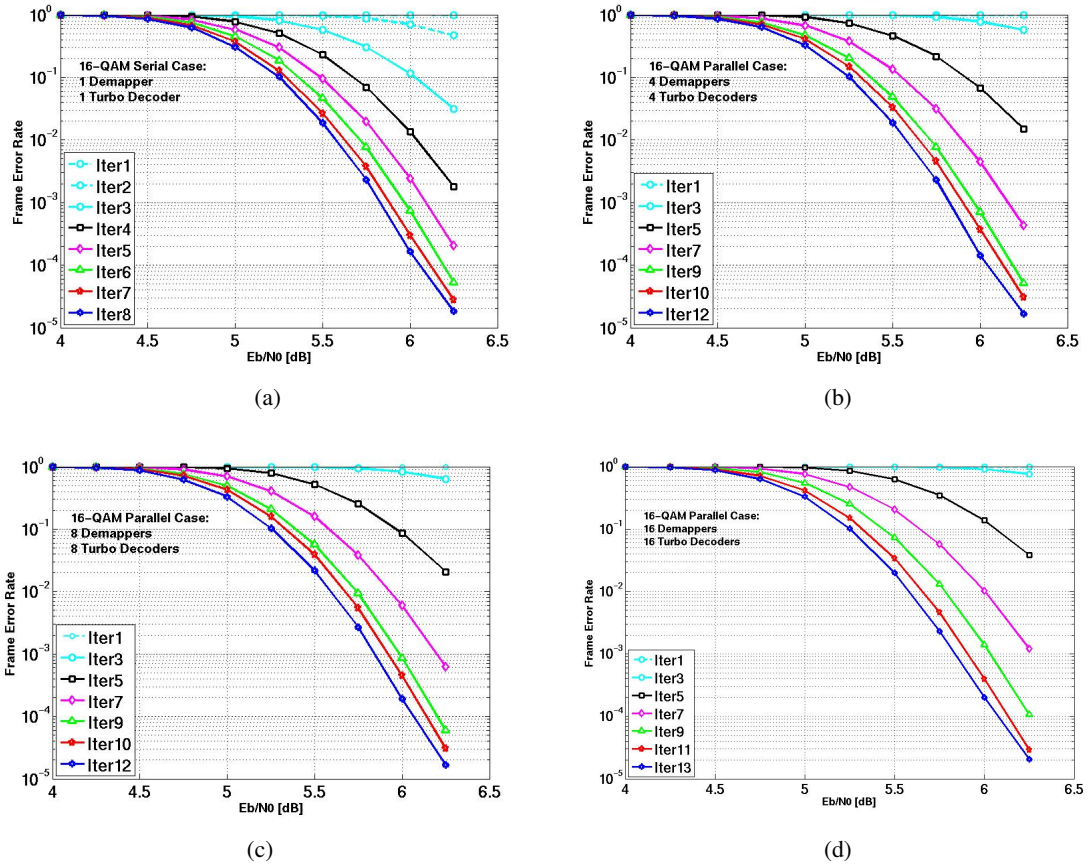


Figure 2.12 — Serial vs Parallel Turbo Demodulation for 188 Source Byte, double binary encoder ,$\frac{1}{2}$ code rate, 16-QAM, Rayleigh fading channel (a) Serial turbo Demodulation ; (b) Parallel turbo demodulation with 4 Demappers 8 Decoders (4 Turbo Decoders); (c) Parallel turbo demodulation with 8 Demappers 16 Decoders (8 Turbo Decoders); and, (d) Parallel turbo demodulation with 16 Demappers 32 Decoders (16 Turbo Decoders).

2.5.1.2 Simulation Results

Different simulations programs were executed for the serial and parallel models of turbo demodulation. For a system with 188-bytes of data at $r = 0.5$, with 16-QAM and 256-QAM constellations, $\alpha = 22.5°$, $d = 1$ and $tp_r = 1$, the results of serial turbo demodulation are shown in Fig. 2.12(a) & 2.13(a) whereas parallel turbo demodulation results are shown in Fig. 2.12(b,c,d) & 2.13(b,c,d). In Fig. 2.13(c)&2.13(d) the results of unbalanced demapping and decoding components are shown.

These system parameters (including the rotation angle) are same as opted in [10] for serial turbo demodulation to have a comparison reference. However, the work in [10] considers original optimal algorithms for decoding (MAP) and demapping (ML) which imply a gain of 0.2 dB as compared to the considered algorithms with max-log simplification.

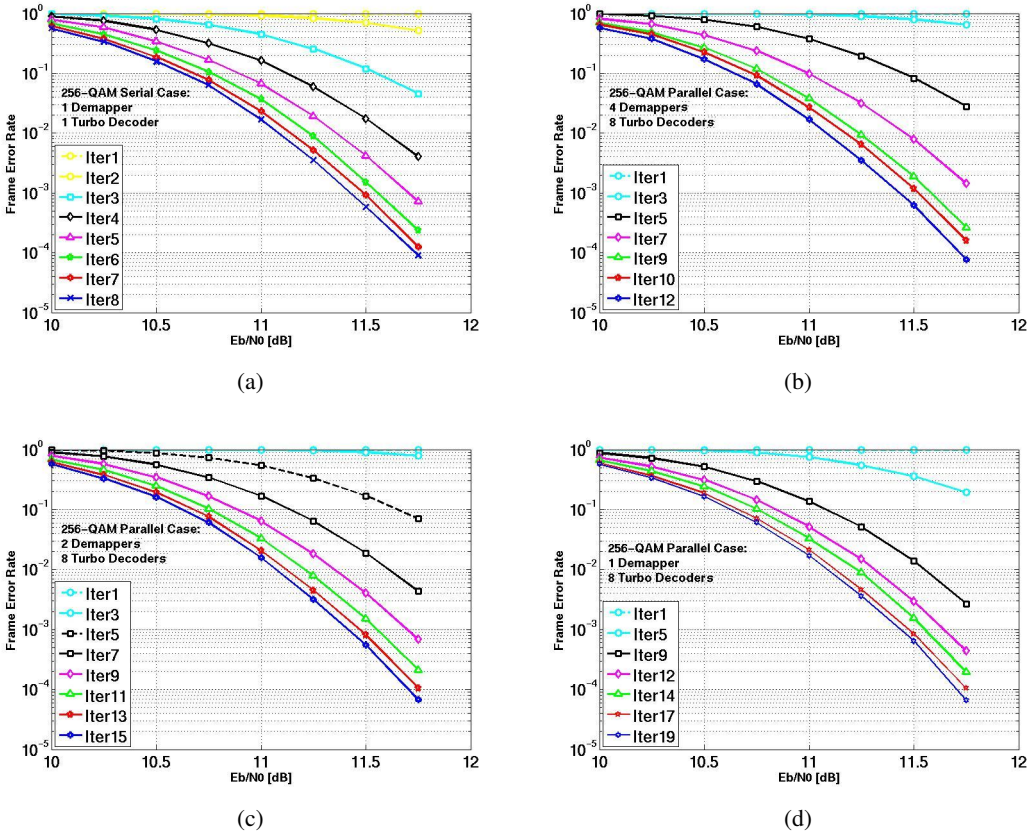


Figure 2.13 — Serial vs Parallel turbo demodulation for for 188 Source Byte, double binary encoder ,$\frac{1}{2}$ code rate, 256-QAM, Rayleigh fading channel (a) Serial turbo demodulation ; (b) Parallel turbo demodulation with 4 Demappers 16 Decoders (8 Turbo Decoders); (c) Parallel turbo demodulation with 2 Demappers 16 Decoders (8 Turbo Decoders); and, (d) Parallel turbo demodulation with 1 Demappers 16 Decoders (8 Turbo Decoders).

To establish the overall performance of the parallel turbo demodulation system two metrics, the speed gain which is the ratio of the serial and parallel execution times and the area overhead which can be expressed in terms of the ratio of hardware used for the parallel and serial systems, are summarized in Table 2.1. A_{dem} and A_{dec} are defined as areas of one component demapper and one component decoder. A_{CN1} is the area of communication interconnect between demappers and turbo decoder, whereas A_{CN2} is the area of communication interconnect between decoder components of the turbo decoder as shown in Fig. 2.11. T units of time is taken as a reference which is the decoding time of a frame in natural or interleaved domain for the serial execution case. In a balanced scenario, for 16-QAM the DD_r is unity, hence an equal number of demapper components and decoder components on each side of the turbo decoder are required. In 256-QAM this ratio is 2.

Considering the first system configuration of 16-QAM for which serial execution is shown in Fig.2.12(a) and one of the parallel execution scenario with 4 demappers and 4 turbo decoders (8 decoder components) is shown in Fig.2.12(b). In parallel case, 12 shuffled iterations will be required to reach the same FER performance of 8 serial turbo demodulations iterations. Regarding serial execution time, the 8 iterations of turbo decodings will take $16T$ units of time. For double binary encoding at $r = 0.5$ and 16-QAM modulation, the number of modulated symbols in the frame will be same as the number of encoded symbol. Hence, with a $tp_r = 1$, demapping will also take T units of time to demap the frame. This will result in $24T$ units of time for 8 turbo demodulation iterations. For

the parallel case with 4 demappers and 4 turbo decoders, first demapping will take $\frac{T}{4}$ units of time and each of the 12 shuffled iterations will take $\frac{T}{4}$ units of time. This will result in a total time of $\frac{13T}{4} = 3.25T$ units of time for parallel turbo demodulation. The resultant time gain is 7.38 and area overhead is $\frac{(4A_{dem}+8A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$. With a reasonable assumption of $A_{CN_{1\&2}} \leq 4A_{dem}$ in area overhead expression for this configuration, the resultant value will be less than equal to 8. Hence, with this assumption the parallelism efficiency (ratio of the time gain and area overhead) approaches to unity. On the same bases, using time gain and area overhead presented in Table 2.1 for other system scenarios, parallelism efficiency can be computed.

An interesting aspect shown in Table 2.1 is the marginal increased in number of shuffled iterations when the parallelism level is increased in a balanced system. This is due to the fact that if *a priori* information related to even a single bit of a modulated symbol is sent to the demapper by the decoder, all the bits related to the symbol will be updated by the demapper and hence will help in rapid convergence. This leads to a linear increase in speed gain with parallelism degree for balanced systems. However, for unbalanced system more iterations are required as illustrated in the last two rows of Table 2.1).

Using the expressions of the last column of Table 2.1 and the area information of the SISO components, a hardware designer can estimate the parallelism efficiency of an architecture to tradeoff between throughput and area overhead.

Table 2.1 — Parallelization efficiency results

188 Byte Source DVB-RCS Encoder Random $\prod_2$ Rayleigh Fading Channel							
16-QAM $\alpha = 22.5°$ $r = 0.5$ $tp_r = 1$ $DD_r = 1$							
Serial Execution							
Ref.	Hardware Resources	Demapping Time	Decoding Time	No. of Iterations	Total Time		
Fig.2.12(a)	1 Dem 1 Dec	T	T	8	$24T$		
Parallel Execution							
Ref.	Hardware Resources	1^{st}Demapping Time	Shuffling iteration Time	No. of shuffling iterations	Total Time	Speed Gain	Area Overhead
Fig.2.12(b)	4 Dem 8 Dec 2 CN	$\frac{T}{4}$	$\frac{T}{4}$	12	$\frac{13T}{4}$	7.38	$\frac{(4A_{dem}+8A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$
Fig.2.12(c)	8 Dem 16 Dec 2 CN	$\frac{T}{8}$	$\frac{T}{8}$	12	$\frac{13T}{8}$	14.77	$\frac{(8A_{dem}+16A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$
Fig.2.12(d)	16 Dem 32 Dec 2 CN	$\frac{T}{16}$	$\frac{T}{16}$	13	$\frac{14T}{16}$	27.43	$\frac{(16A_{dem}+32A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$
256-QAM $\alpha = 22.5°$ $r = 0.5$ $tp_r = 1$ $DD_r = 2$							
Serial Execution							
Ref.	Hardware Resources	Demapping Time	Decoding Time	No. of Iterations	Total Time		
Fig.2.13(a)	1 Dem 1 Dec	$\frac{T}{2}$	T	8	20T		
Parallel Execution							
Ref.	Hardware Resources	1^{st}Demapping Time	Shuffling iteration Time	No. of shuffling Iterations	Total Time	Speed Gain	Area Overhead
Fig.2.13(b)	4 Dem 16 Dec 2 CN	$\frac{T}{8}$	$\frac{T}{8}$	12	$\frac{13T}{8}$	12.3	$\frac{(4A_{dem}+16A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$
Fig.2.13(c)	2 Dem 16 Dec 2 CN	$\frac{T}{4}$	$\frac{T}{8}$	15	$\frac{17T}{8}$	9.41	$\frac{(2A_{dem}+16A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$
Fig.2.13(d)	1 Dem 16 Dec 2 CN	$\frac{T}{2}$	$\frac{T}{8}$	19	$\frac{21T}{8}$	7.619	$\frac{(A_{dem}+16A_{dec}+A_{CN1}+A_{CN2})}{(A_{dem}+A_{dec})}$

2.5.2 Parallel Turbo Equalization

A software model implementing a parallel turbo equalization was created in C++ programming language. On the transmitter side, double binary turbo code, BICM interleaver, mapper and MIMO Spatial Multiplexing (SM) of Wimax standard are modeled. The channel is modeled as MIMO Rayleigh fading.

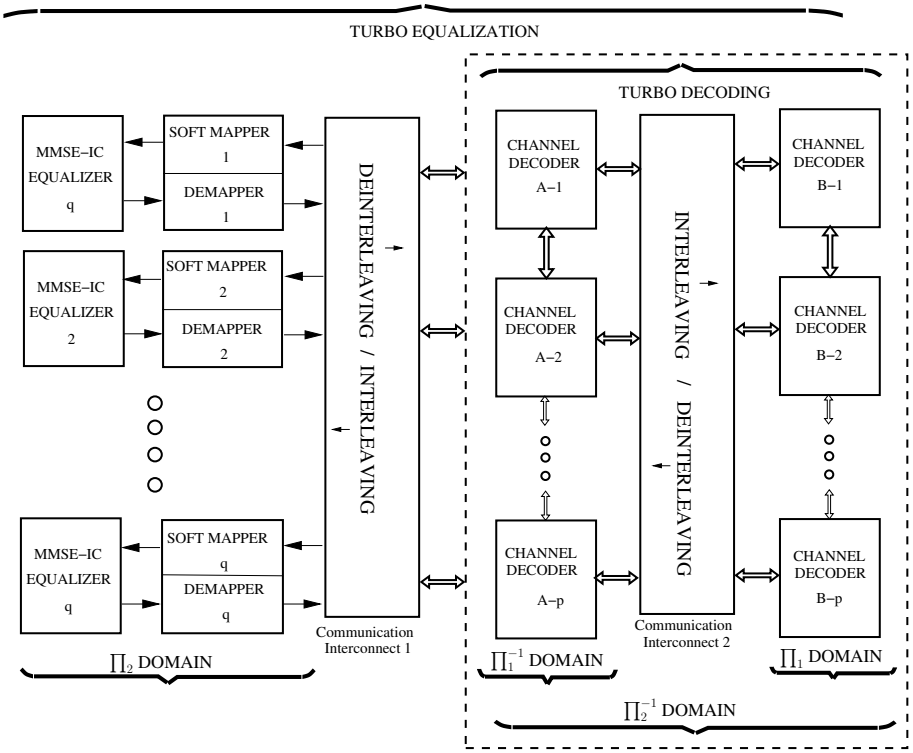


Figure 2.14 — Architecture of receiver's software model

2.5.2.1 Software Model for Parallel Turbo Equalization

The software architectural diagram of the parallel turbo equalizer is shown in Fig. 2.14. Based on the same concept as used in parallel turbo demodulation, consider an example of a frame having A source symbols where each symbol is made up of n bits, encoded at a code rate r, then modulated with m bits per symbol and finally R symbols are transmitted per STC transmission time on MIMO channel. In this case, there will be $B = \frac{A \times n}{r \times m \times R}$ modulated symbol vectors in the frame. In a shuffled turbo equalization system, the best case is when the receiver is balanced in a way that both decoding and equalization tasks finish at the same time. Demapping and soft mapping are implicitly included in the equalization task due to their pipelining in parallel execution. Since the equalization and decoding tasks are different in nature, another parameter, the symbol processing throughput ratio of decoder and equalizer component "$tp_r = \frac{Decoder\ throughput}{Equalizer\ throughput}$" is required. In this scenario, the ratio of decoder and equalizer components (DE_r), required to finish the frame at the same time can be deduced as follows:

$$\text{Time for decoding} = \text{Time for equalization} \tag{2.53}$$

$$\frac{\frac{A}{No.\ of\ Decoders\ (p)}}{Decoder\ throughput} = \frac{\frac{B}{No.\ of\ Equalizers\ (q)}}{Equalizer\ throughput} \tag{2.54}$$

$$DE_r = \frac{No.\ of\ Decoders\ (p)}{No.\ of\ Equalizers\ (q)} = \frac{A}{B \times tp_r} = \frac{r \times m \times R}{n \times tp_r} \tag{2.55}$$

Using the above expression, for a dual binary turbo encoder with code rate $r = 0.5$, 16-QAM 2×2 spatially multiplexed system ($R = 2$), and $tp_r = 1$ the parameter $DE_r = 2$, i.e, for one equalizer component two decoder components on each side of the shuffled turbo decoder are required.

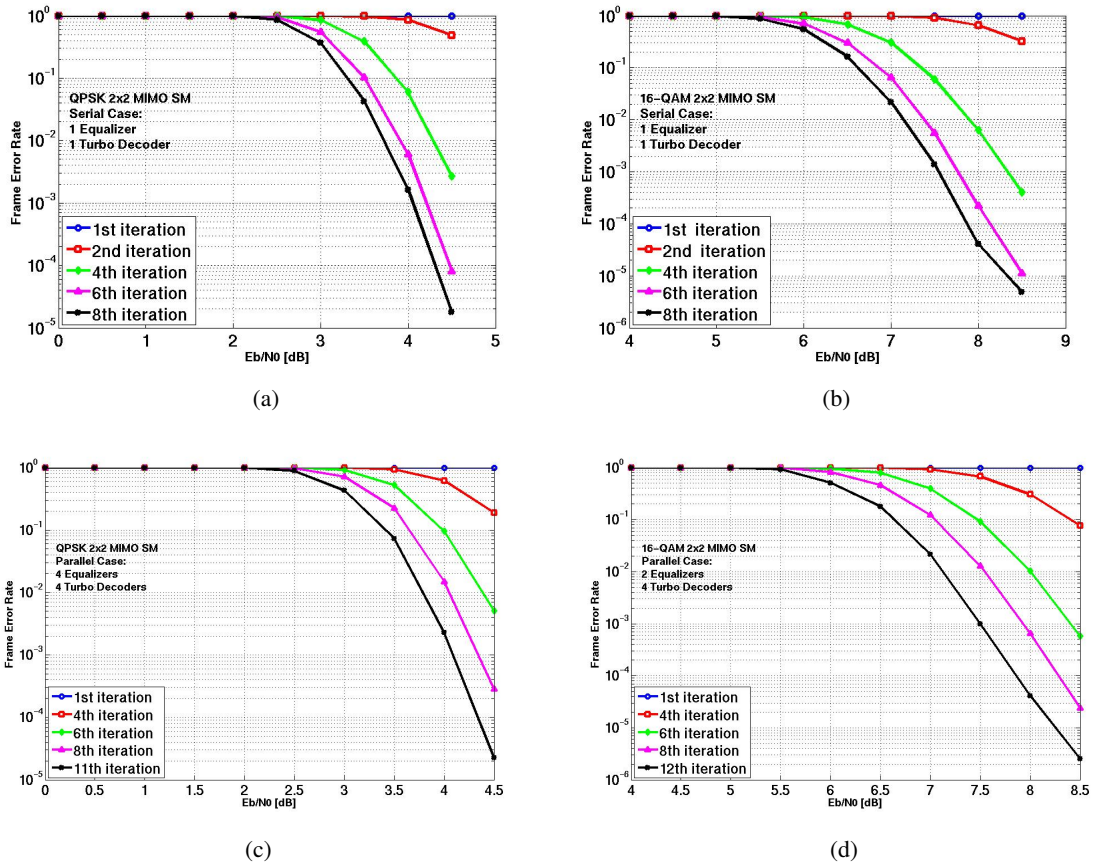


Figure 2.15 — Serial vs parallel turbo equalization for 120 Source Byte, double binary encoder , $\frac{1}{2}$ code rate, $2\times$ 2 MIMO SM (a) QPSK Serial turbo equalization ; (b) 16-QAM Serial turbo equalization; (c) QPSK parallel turbo equalization with 4 Equalizers 8 Decoders (4 Turbo Decoders) $tp_r = 1$; and, (d) 16-QAM Parallel turbo equalization with 2 Equalizers 8 Decoders (4 Turbo Decoders) $tp_r = 1$.

2.5.2.2 Simulation Results

For a system transmitting frames of 120 bytes at $r = 0.5$ using QPSK and 16-QAM, 2×2 and 4×4 MIMO SM through Fast Rayleigh fading channel, Frame Error Rate (FER) results of serial and parallel turbo equalization are shown in Fig. 2.15 and Fig. 2.16.

Considering the first system configuration of QPSK 2×2 MIMO SM (Fig. 2.15(a) for serial turbo equalization and Fig. 2.15(c) for parallel turbo equalization), 11 shuffled iterations using 4 equalizers and 4 turbo decoders (8 decoder components) at $tp_r = 1$ provide the same FER performance as 8 serial iterations. Regarding execution time in serial case, if T time units are taken by one component decoder to process a frame, the time consumed by a turbo decoder will be $2T$ per iteration. For double binary encoding at $r = 0.5$, QPSK modulation and 2×2 MIMO SM, the number of modulated symbols vectors in the frame will be same as the number of encoded symbol. Hence, with a

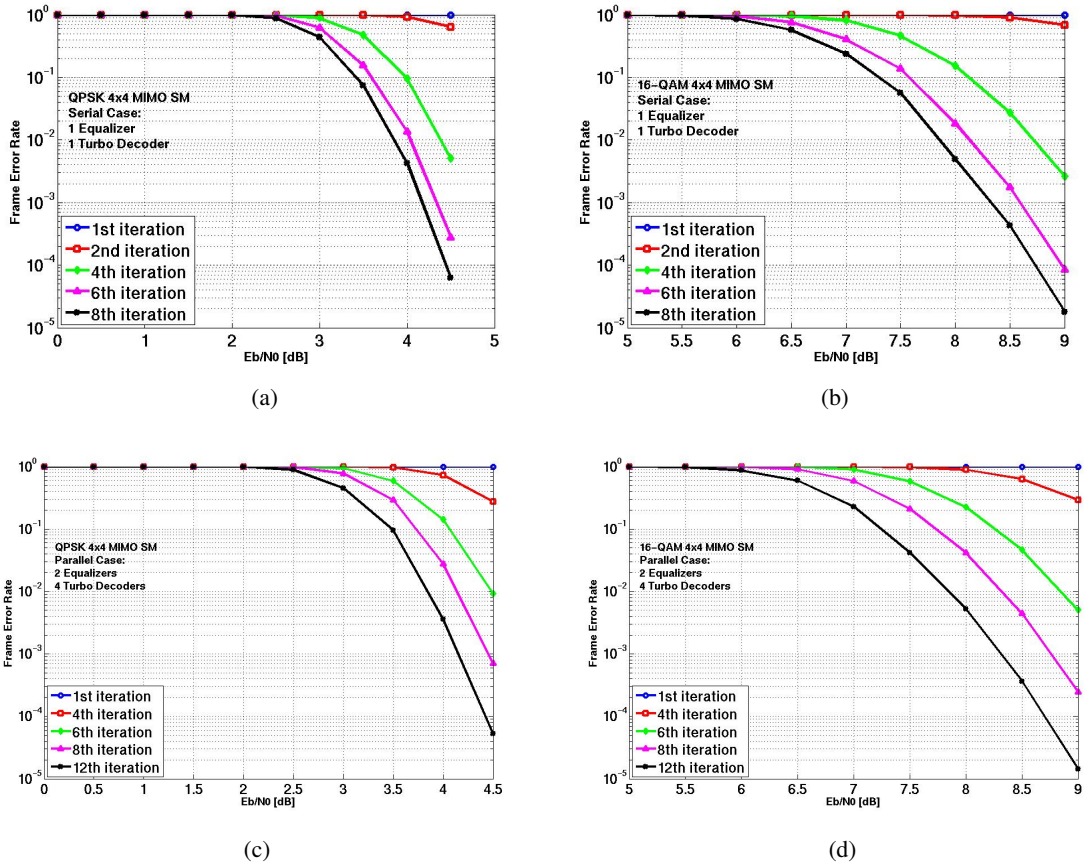


Figure 2.16 — Serial vs parallel turbo equalization for 120 Source Byte, double binary encoder ,$\frac{1}{2}$ code rate, $4\times$ 4 MIMO SM (a) QPSK Serial Turbo equalization ; (b) 16-QAM Serial Turbo equalization; (c) QPSK Parallel turbo equalization with 2 Equalizers 8 Decoders (4 Turbo Decoders) $tp_r = 1$; and, (d) 16-QAM Parallel Turbo equalization with 2 Equalizers 8 Decoders (4 Turbo Decoders) $tp_r = 2$.

$tp_r = 1$, equalization and demapping tasks (where demapping is performed in pipelined way with equalization) will also take T units of time. Soft mapping will also consume T time units per iteration in order to match its throughput with equalizer and demapper in parallel case. Since in the first iteration soft mapping is not required, the total consumed time in serial execution case becomes $31T$ for 8 global iterations. In parallel execution case, using 4 equalizers and 4 turbo decoders with shuffled turbo equalization, the first equalization and demapping pipelined process will take $0.25T$ unit of time and the 11 shuffled iterations will take $11 \times 0.25T$ unit of time. This results in $3T$ unit of time for parallel execution. Hence it gives a speed gain (defined as the ratio of the serial and parallel execution times) equals to 10.3. The area overhead which is the ratio of area of the parallel and serial cases, are summarized in Table 2.2 where A_{equ} and A_{dec} are defined as areas of one equalizer and one decoder component respectively. A_{CN1} is the area of communication interconnect between demappers/soft mappers and turbo decoder, whereas A_{CN2} is the area of communication interconnect between decoder components of the turbo decoder as shown in Fig. 2.14. With a reasonable assumption of $A_{CN_{1\&2}} \leq 4A_{equ}$, parallelism efficiency (ratio of speed gain and area overhead) becomes greater than 1.25. This parallel efficiency of more than unity can be translated as increased convergence in parallel execution case. This comes from the fact that an updated *a posteriori* information related to a single bit of a symbol in a vector, coming from the turbo decoder, will update LLRs related to all bits of the vector at the demapper output. Results of the other system configurations are summarized in Table 2.2 and all of them lead to promising parallelism performance.

Table 2.2 — Parallelization Efficiency Results

120 Byte Source Double Binary Encoder and $\prod_2$ of Wimax Standard 2×2 MIMO SM							
QPSK and 16-QAM $r = 0.5$ $tp_r = 1$							
Serial Execution							
Ref.	Hardware Resources	Equalization Time	Decoding Time	Soft Mapping Time	No. of Iterations	Total Time	
Fig.2.15(a)	1 Equ 1 Dec	T	T	T	8	$31T$	
Fig.2.15(b)	1 Equ 1 Dec	$\frac{T}{2}$	T	$\frac{T}{2}$	8	$23.5T$	
Parallel Execution							
Ref.	Hardware Resources	1^{st}Demapping Time	Shuffling iteration Time	No. of shuffling iterations	Total Time	Speed Gain	Area Overhead
Fig.2.15(c)	4 Equ 8 Dec 2 CN	$\frac{T}{4}$	$\frac{T}{4}$	11	$\frac{12T}{4}$	10.3	$\frac{(4A_{equ}+8A_{dec}+A_{CN1}+A_{CN2})}{(A_{equ}+A_{dec})}$
Fig.2.15(d)	2 Equ 8 Dec 2 CN	$\frac{T}{4}$	$\frac{T}{4}$	11	$\frac{12T}{4}$	7.8	$\frac{(2A_{equ}+8A_{dec}+A_{CN1}+A_{CN2})}{(A_{equ}+A_{dec})}$
120 Byte Source Double Binary Encoder and $\prod_2$ of Wimax Standard 4×4 MIMO SM							
QPSK and 16-QAM $r = 0.5$ $tp_r = 1$ for QPSK and $tp_r = 2$ for 16-QAM							
Serial Execution							
Ref.	Hardware Resources	Equalization Time	Decoding Time	Soft Mapping Time	No. of Iterations	Total Time	
Fig.2.16(a)	1 Equ 1 Dec	$\frac{T}{2}$	T	$\frac{T}{2}$	8	$23.5T$	
Fig.2.16(b)	1 Equ 1 Dec	$\frac{T}{2}$	T	$\frac{T}{2}$	8	$23.5T$	
Parallel Execution							
Ref.	Hardware Resources	1^{st}Demapping Time	Shuffling iteration Time	No. of shuffling iterations	Total Time	Speed Gain	Area Overhead
Fig.2.16(c)	2 Equ 8 Dec 2 CN	$\frac{T}{4}$	$\frac{T}{4}$	12	$\frac{13T}{4}$	7.2	$\frac{(2A_{equ}+8A_{dec}+A_{CN1}+A_{CN2})}{(A_{equ}+A_{dec})}$
Fig.2.16(d)	2 Equ 8 Dec 2 CN	$\frac{T}{4}$	$\frac{T}{2}$	12	$\frac{14T}{4}$	6.7	$\frac{(2A_{equ}+8A_{dec}+A_{CN1}+A_{CN2})}{(A_{equ}+A_{dec})}$

2.6 Conclusion

In this chapter, we have summarized three SISO algorithms related to the equalizer, demapper and convolutional turbo decoder blocks of an iterative receiver. Simplified expressions of the considered algorithms, suitable for hardware implementations, are also provided.

To address the issues of latency and low throughout, associated with iterative receivers, parallelism in turbo decoder is recalled from previous works. For turbo demodulation and turbo equalization, parallelism on three different level is proposed in this chapter. At the end of the chapter, through the software model results, speed gain and parallelism efficiency for parallel turbo demodulation and parallel turbo equalization are presented and analyzed for different system configurations. It is also shown that for a balanced system where different tasks finish together, shuffled turbo demodulation/equalization provides the benefits for convergence. In addition, the presented parallel modeling gives the user an estimate on hardware cost and parallelism efficiency for the selection of an optimized parallelism degree.

CHAPTER

3 Heterogeneous Multi-ASIP NoC-based Approach

WHILE the first two chapters have addressed the algorithmic aspects of the target flexible high-throughput iterative receiver, this chapter introduces the explored implementation approach.

Flexibility and high-throughput requirements are being widely investigated in the design of digital receivers during the last few years. Several implementations have been proposed. Some of these implementations succeeded in achieving high throughput for specific standards with a highly dedicated architecture. However, these implementations do not take into account flexibility and scalability issues. Conversely, others implementations include software and/or reconfigurable parts to achieve the required flexibility while achieving much lower throughput.

In this PhD work our aim is to tackle flexibility and performance requirements simultaneously by proposing multiprocessor architectures. The architecture models we are exploring are based on Application-Specific Instruction-set Processors (ASIP) interacting through an adequate communication network in a multi-ASIP architecture platform.

The first part of this chapter introduces the evaluation of embedded processor architectures towards customizable instruction-set ones. This crucial efficiency-driven evolution constitutes our main motivation behind the selection of application-specific instruction-set (ASIP) design approach. Section x gives an overview on existing ASIP design flows and presents the considered CoWare's design tool: Processor Designer.In the second part of the chapter, Network on Chip (NOC) is presented as a solution for the communication required between SISO components in a parallel turbo receiver. Finally, at the end of the chapter a multi-ASIP and NOC based flexible, scalable and parallel turbo receiver architecture has been proposed which includes information about already conceived multi-ASIP and NOC based turbo receiver.

3.1 Customizable Embedded Processors

The complexity of a large share of the integrated circuits manufactured today is impressive [45]: devices with hundreds of millions of transistors are not uncommon. Unsurprisingly, the non-recurrent engineering costs of such high-end application-specific integrated circuits is approaching a hundred million U.S. dollars-a cost hardly bearable by many products individually. It is mainly the need to increase the flexibility and the opportunities for modular reuse that is pushing industry to use more and more software-programmable solutions for practically every class of devices and applications.

On the other hand, processor architecture has evolved dramatically in the last couple of decades [45]: from microprogrammed finite state machines, processors have transformed into single rigid pipelines; then, they became parallel pipelines so that various instructions could be issued at once; next, to exploit the ever-increasing pipelines, instructions started to get reordered dynamically; and, more recently, instructions from multiple threads of executions have been mixed into the pipelines of a single processor, executed at once. However, now something completely different is changing in the lives of these devices: on the whole, the great majority of the high-performance processors produced today address relatively narrow classes of applications. This is related to one of the most fundamental trends that slowly emerged in the last decade: to design tailor-fit processors to the very needs of the application rather than to treat them as rigid fixed entities, which designers include as they are in their products. The emergence of this trend has been made successful thanks to the development of new adequate design methodologies and tools. Such tools enable designers to specify a customizable processor, and in some cases completely design one, in weeks rather than months. Leading companies in providing such methodologies and tools include CoWare (acquired recently by Synopsys), Tensilica, ARC Cores, Hewlett-Packard, and STMicroelectronics. The shape and boundaries of the architectural space covered by the tool chain differentiate the several approaches attempted. Roughly, these approaches can be classified in three categories [45]:

Parameterizable processors are families of processors belonging to a single family and sharing a single architectural skeleton, but in which some of the characteristics can be turned on or off (presence of multipliers, of floating point units, of memory units, and so forth) and others can be scaled (main datapath width, number and type of execution pipelines, number of registers, and so forth).

Extensible processors are processors with some support for application-specific extensions. The support comes both in terms of hardware interfaces and conventions and in terms of adaptability of the tool chain. The extensions possible are often in the form of additional instructions and corresponding functional pipelines but can also include application-specific register files or memory interfaces.

Custom processor development tools are frameworks to support architects in the effort to design from scratch (or, more likely, from simple and/or classic templates) a completely custom processor with its complete tool chain (compiler, simulator, and so forth). Ideally, from a single description in a rich architectural description language (ADL), all tools and the synthesizable description of the desired core can be generated.

It is worth noting that these approaches are not mutually exclusive: A parameterizable processor may also be extensible. A template processor in a processor development framework can be easily parameterized and is naturally extended. All these approaches fall under the name of customizable processors and often are referred as ASIP for Application-Specific Instruction-set Processors.

3.2 ASIP Design

Application Specific Instruction-set Processors (ASIPs) are increasingly used in complex System on Chip (SoC) designs. ASIPs are tailored to particular applications, thereby combining performance and energy efficiency of dedicated hardware solutions with the flexibility of a programmable solution. The main idea is to design a programmable architecture tailored to a specific application, thus preserving a much higher degree of flexibility than a dedicated ASIC solution.

3.2.1 Design flow overview

Typically, the development flow of ASIPs starts with the analysis of the application in order to identify its "hot spots" [46]. Then, an initial architecture is defined, in particular with special custom instructions to improve the efficiency for handling those hot spots. Afterwards the applications are run on the processor in order to verify if the target specifications have been met. If that is not the case, the whole flow is iterated to meet the design requirements for given applications.

From this quick overview of the design flow it is clear that some tools are required for implementing it: first, an assembler and a linker are needed in order to run the application code on the processor, together with a compiler if a high-level programming language is used; these tools are required both for design space exploration, when the target application has to be tested in order to improve the architecture, and for software development after the final architecture has been defined. Moreover, an Instruction Set Simulator (ISS) has to be provided so that the application can be run both for profiling and for verification purposes. All these tools directly depend on the instruction set of the processor and hence they have to be retargeted each time that the instruction set is modified. Almost all the available ASIP design suites provide these tools and the capability to retarget them when needed, while some of them also include the further ability to automate the process of profiling the application and identifying the instructions which are most suitable for instruction-set extension.

By looking at available commercial solutions for ASIP design , it is possible to identify three main classes based on the degree of freedom which is left to the designer [46]:

- Architecture Description Language (ADL) based solutions (e.g. CoWare Processor Designer [47], Target IP Designer [48]), which can be also defined as ASIP-from-scratch since every detail of the architecture, including for instance pipeline and memory structures, can be accessed and specified by the designer by means of a proper language. This approach results in the highest flexibility and efficiency, but on the other hand it requires a significant design effort.
- Template architecture based solutions (e.g. Tensilica Xtensa [49], ARC ARChitect [50]), which allow the designer to add custom ISE to a pre-defined and pre-verified core, thus restricting the degree of freedom with respect to the previous approach to the instruction set definition only.
- Software configurable processors and reconfigurable processors (e.g. Stretch [51], ADRES [52]), with a fixed hardware including a specific reconfigurable ISE fabric which allows the designer to build custom instructions after the fabrication.

3.2.2 CoWare's ADL-based design tool: Processor Designer

CoWare Processor Designer is an ASIP design environment entirely based on LISA [53]. The language syntax provides a high flexibility to describe the instruction set of various processors, such as SIMD (Single-Instruction Multiple-Data), MIMD (Multiple-Instruction Multiple-Data) and VLIW

(Very long instruction word)-type architectures. Moreover, processors with complex pipelines can be easily modeled.

Processor Designer's high degree of automation greatly reduces the time for developing the software tool suite and hardware implementation of the processor, which enables designers to focus on architecture exploration and development. The usage of a centralized description of the processor architecture ensures the consistency of the Instruction-Set Simulator (ISS), software development tools (compiler, assembler, and linker etc.) and RTL (Register Transfer Level) implementation, minimizing the verification and debug effort.

The LISA machine description provides information consisting of the following model components [46]:

- The memory model lists the registers and memories of the system with their respective bit widths, ranges and aliasing.
- The resource model describes the available hardware resources, like registers, and the resource requirements of operations. Resources reproduce properties of hardware structures which can be accessed exclusively by a given number of operations at a time.
- The instruction set model identifies valid combinations of hardware operations and admissible operands. It is expressed by the assembly syntax, instruction word coding, and the specification of legal operands and addressing modes for each instruction.
- The behavioral model abstracts the activities of hardware structures to operations changing the state of the processor for simulation purposes. The abstraction level can range widely between the hardware implementation level and the level of high-level language (HLL) statements.
- The timing model specifies the activation sequence of hardware operations and units.
- The micro-architecture model allows grouping of hardware operations to functional units and contains the exact micro-architecture implementation of structural components such as adders, multipliers, etc.

By using these various model components to describe the architecture, it is then possible to generate a synthesizable HDL representation and the complete software tool suite automatically.

The generation of the software development environment by Processor designer enables to start application software development prior to silicon availability, thus eliminating a common bottleneck in embedded system development. As it is shown in Fig.3.1, the design flow of Processor Designer is a closed-loop of architecture exploration for the input applications. It starts from a LISA 2.0 description, which incorporates all necessary processor-specific components such as register files, pipelines, pins, memory and caches, and instructions, so that the designer can fully specify the processor architecture. Through Processor Designer, the ISS and the complete tool suite (C-compiler, assembler, linker) are automatically generated. Simulation is then run on the architecture simulator and the performance can be analyzed to check whether the design metrics are fulfilled. If not, architecture specifications are modified in LISA description until design goals are met. At the end, the final version of RTL implementation (Verilog HDL, VHDL and SystemC) together with software tools is automatically generated.

As previously mentioned, ASIPs are often employed as basic components of more complex systems, e.g. MPSoCs. Therefore, it is very important that their design can be embedded into the overall system design. Processor Designer provides possibilities to generate a SystemC model for the processor, so that it can be integrated into a virtual platform. In this way, the interaction of the processor

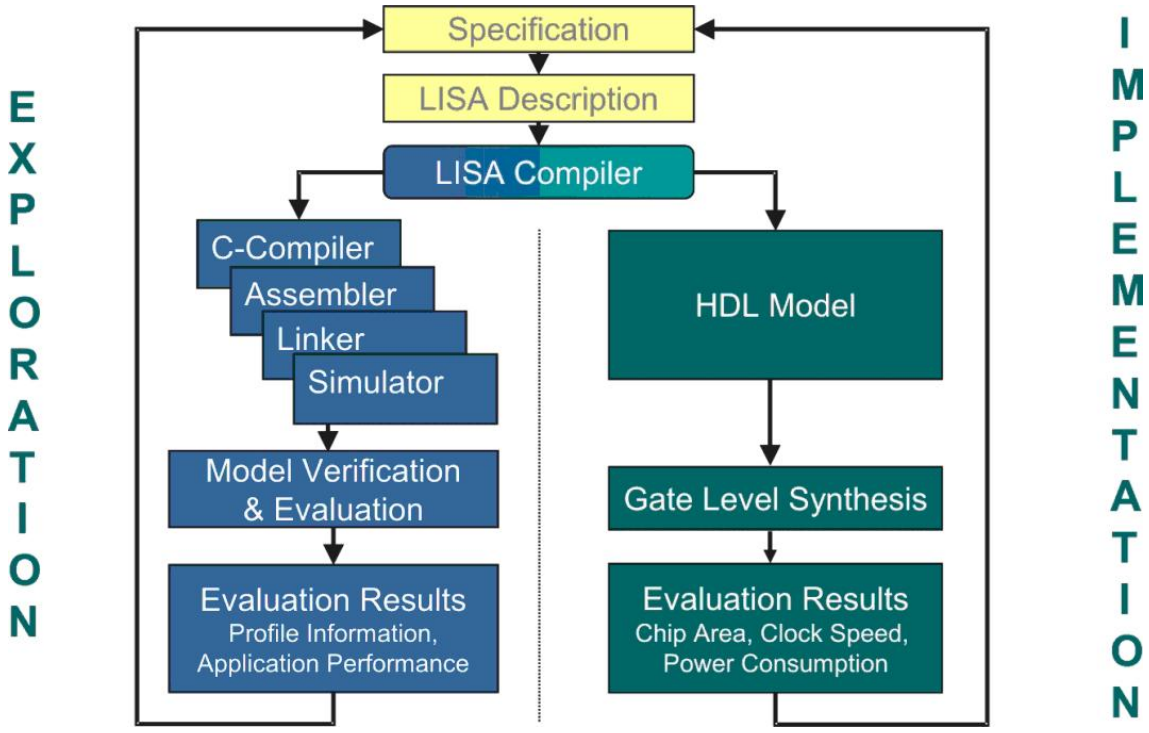


Figure 3.1 — LISA architecture exploration flow

with the other components in the system can be tested. Furthermore, the exploration as well as the software development of the platform at early design stage becomes possible.

3.3 NoC as communication interconnect

Besides application algorithm optimizations and application-specific processor design, the on-chip communication network connecting the multiple on-chip cores constitutes a major issue. Conventional on-chip buses become inefficient in large systems and the nanotechnology integration issues (propagation delay, crosstalk, etc.) make their use no more practical. In this context, Network-on-Chip has recently emerged as a new paradigm allowing to cope with these major design issues. It consists of adapting the modular, scalable, and flexible hardware/software architectures and design tools of Network domain to the context of silicon integration.

In this thesis work Network on Chip (NoC) paradigm is considered for conflict free information exchange between ASIPs. Although this work reutilizes one of the NoC architecture proposed in [43], for the sake of continuity, some background about NoC and its application in the field of digital communication applications are presented below.

3.3.1 Emergence of InterIP-NoC

As adequately summarized in [54], after the introduction of the NoC concept [55, 56, 57, 58] the scientific literature focused on general purpose NoC solutions. More recently the idea of Application Specific NoC (ASNoC) [59] was proposed as a method to improve efficiency, through a careful tailoring of the network features around the specific application to be supported. All ASNOC examples

available in the literature are related to fairly complex applications, which involve heterogeneous processing tasks or IPs (Intellectual Property), occupy a fairly relevant physical area and make use of advanced methods for routing and congestion control. Following the classification given in [60], we call this kind of NoC InterIP-NoC. On the contrary, [60] defines as IntraIP-NoC any network whose domain area is restricted to be internal to a single IP: in this case, the NoC extend is typically much smaller, processing tasks are usually homogeneous and stringent constraints are posed on the NoC overhead, which is limited resorting to simple routing methods.

One of the most recently investigated application for this emerging concept is turbo decoding [43, 54]. In the context of a parallel multiprocessor turbo decoder with capability of supporting generic interleavers with no collisions, the NoC (or better IntraIP-NOC [60]) is a very interesting option for the implementation of the flexible interconnect structure. NoCs bring to this specific kind of application several of their general advantages over traditional on-chip interconnects, such as enhanced scalability, separation between computation and communication, modularity, regularity of physical links and predictability of their delay [54].

3.3.2 Network Topologies and Routing

Network topologies can be classified as direct and indirect networks [54]. Direct Networks typically consists of a set of nodes, each one being directly connected to small subset of other nodes in the network. The required connectivity is obtained by means of routers, which are components of the nodes and can decide the path for each data to be sent from a source node to a destination node. Instead of providing a direct connection between two nodes, indirect networks exploit switches to connects nodes. Few typical network topologies used in literature for NoC design are presented in Figure 1.5 [61].

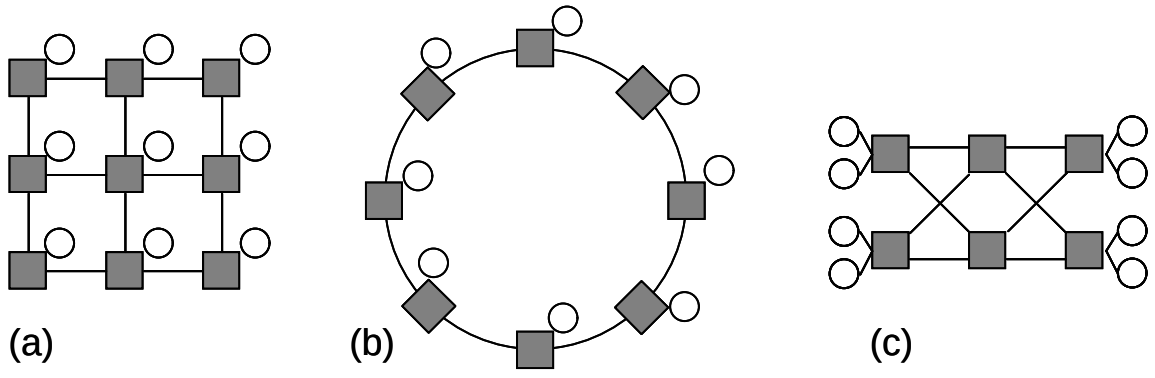


Figure 3.2 — NoC topologies (a) 2D-mesh direct topology (b) Ring direct topology (c) Multistage indirect topology

The mesh-like topologies are most common in the literature, mainly due to multiple benefits provided by their regularity (routing geometry and limited connectivity) [62]. According to need of average distance and connectivity, the mesh topology can be transformed in several forms (classical mesh as in Fig.3.2.a, cylinder, ...) and in several dimensions (line, plane, cube ...). Due to direct connection with neighbors, mesh topologies are well adapted for the applications which require mostly the local data transfer. For the scenario of random communication, network latency mostly depends

on the average distance between two nodes of the topology, which is relatively high for a 2D-mesh topology. The ring topology (Fig.3.2.b) and indirect topologies (Fig.3.2.c), are although somewhat less regular, allow smaller average distances between nodes. Starting with a ring topology, it is possible to reduce the average distance by adding links across the ring. This is called ring topologies with strings. In return, these topological changes involve greater connectivity of the nodes and therefore the resultant routing cost is high. The indirect or multistage networks allow to retain limited connectivity, while maintaining a low average distance [61]. In contrast, these topologies do not allow fast local data transfer as the traffic has to pass through intermediate stages that do not have connected resources.

Overall, choice of a topology for the communication requirements of an application is dictated by performance constraints (throughput, latency), by the constraints of complexity (connectivity, number of nodes and complexity of the nodes), and also by the inherent physical constraints of silicon integration.

Regarding routing, the routers must provide transportation for packets in the network by managing flows and congestion on the network. For this purpose, routing mechanisms, arbitration, flow control, and control information provided with each packet is used. This control information is generally grouped into the header of the packet, after which payload is added i.e the message to convey.

3.3.3 NoC Examples in Iterative Decoding

Communication needs in iterative decoders comes mainly due to transfer of extrinsic information between the two component decoders according to the concerned interleaving rules. In parallel turbo decoding, once sub-blocking and/or shuffled decoding is applied, the induced parallel transfer can rise conflicts when two or more processing units want to write in the same memory bank. The first effort to handle this problem is to take into account parallelism as one of the design parameter while constructing an interleaver without degrading the error rate performance. Multiple publications [63][64][65] address this problem by presenting different types of adequate interleavers for turbo codes. Although these rules of permutation are designed in a way that sub-blocking does not generate a collision and still offering good performance in terms of error correction, it is limited to well-defined codes and certain parallelism degrees.

Flexible interconnection networks capable to support any interleaving rule and high parallelism degree, while featuring low latency and low complexity can constitute a promising solution for this application. In this context, application-specific InterIP-NoCs have been proposed for parallel turbo decoding [43, 54]. Both direct and indirect topologies have been investigated in this domain. A general framework for the design and simulation of NoC-based turbo decoder architectures is proposed in [66]. An architecture based on 2D-mesh topology is presented in [67] for MPSoC turbo and LDPC decoding. Butterfly and Benes networks are presented in [68] for flexible turbo decoders whereas a NoC both for turbo and LDPC is proposed in [69] which is based on the topology of De Brujin Binary Graph.

3.4 Design Approach Illustration: Flexible Parallel Turbo Decoder

In this subsection, we will illustrate the target design approach through the first effort carried out in the Electronics Department of Télćom Bretagne to design a high throughput flexible turbo decoder. To address high throughput requirement, first of all, a comprehensive study was made in exploring the efficient parallelism, at different levels, within a turbo decoder. This parallelism study is detailed

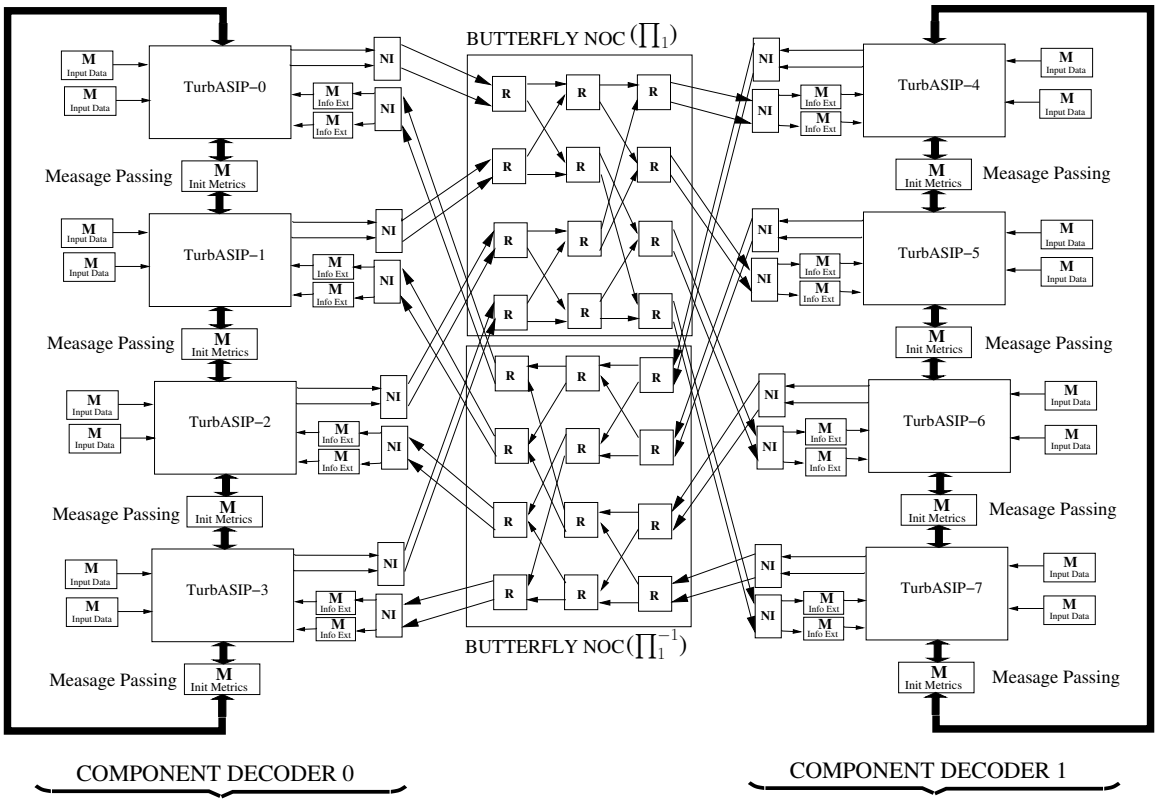


Figure 3.3 — Multi-ASIP and Butterfly NoC architecture for parallel turbo decoder

in [70] and is summarized in subsection 2.4.1 of chapter 2. As the first parallelism level (BCJR metric level) occurring inside a BCJR SISO decoder is the most area efficient, a hardware implementation achieving high throughput should first exploit this parallelism. A dedicated processing architecture with multiple functional units and adequate memory interfaces can be suitable to efficiently perform all computations of a BCJR-SISO decoder. However, as the parallelism degree of this level is limited, hence further increase of throughput should exploit the second parallelism level 2.4.1.2. This can be done efficiently by instantiating multiple processing units with dedicated on-chip communication interconnect.

As the flexibility requirement is also considered, besides high throughput, the processing unit and the on-chip communication should be flexible. A tradeoff between performance and flexibility is thus imposed for the processing unit architecture and the ASIP design approach is thus adopted. Regarding the on-chip communication, appropriate NoC architectures are explored and designed.

In the following subsections, we will illustrate the design approach of the ASIP architecture for turbo decoding, namely TurbASIP, and the designed NoC architecture based on Butterfly topology. Using ASIP and NoC approach, the first 8-ASIP and NoC based turbo decoder [71], implementing first two level of parallelism, is illustrated in Fig.3.3.

3.4.1 TurbASIP

The first step towards the ASIP design for turbo decoding was to extract the flexibility requirements of target standards as summarized in Table 1.1. The complexity of convolutional turbo codes proposed in all existing and emerging standards is limited to eight-state double binary turbo codes or

sixteen-state simple binary turbo codes. Hence, to fully exploit trellis transition parallelism for all standards, a parallelism degree of 32 is required. The implementation of future more complex codes can be supported by splitting trellis sections into sub-sections of 32-parallelism degrees and by processing sub-sections sequentially. Regarding BCJR computation parallelism, a parallelism degree of two has been adopted, i.e. two recursion units to implement the Butterfly metric computation scheme presented in subsection 2.4.1.1. In order to present the TurbASIP architecture, a bottom-up approach is adopted where the basic building blocks are explained first. Based on these building blocks the architecture of recursion units are then presented and finally the full ASIP architecture is illustrated.

3.4.1.1 Building Blocks of TurbASIP

The flexibility parameters of this ASIP are fundamentally based on supporting single and double binary turbo codes implementing the expressions (2.10), (2.12), (2.13) and (2.14) to computed γ, α, β and extrinsic information respectively in logarithmic domain. To achieve this goal the detailed flexibility parameters are derived from these expressions of max-log-MAP algorithm. The detail of building blocks constituting the TurbASIP is briefly discussed below:

Gamma (γ) Metric Computation: As stated in parallelism of trellis transition part of Section 2.4.1.1, all possible values of γ related to all transitions of the trellis are computed in parallel. In hardware this is achieved by the use of simple adders and subtracters which use the input channel LLRs and extrinsic input LLRs to generate γ metrics.

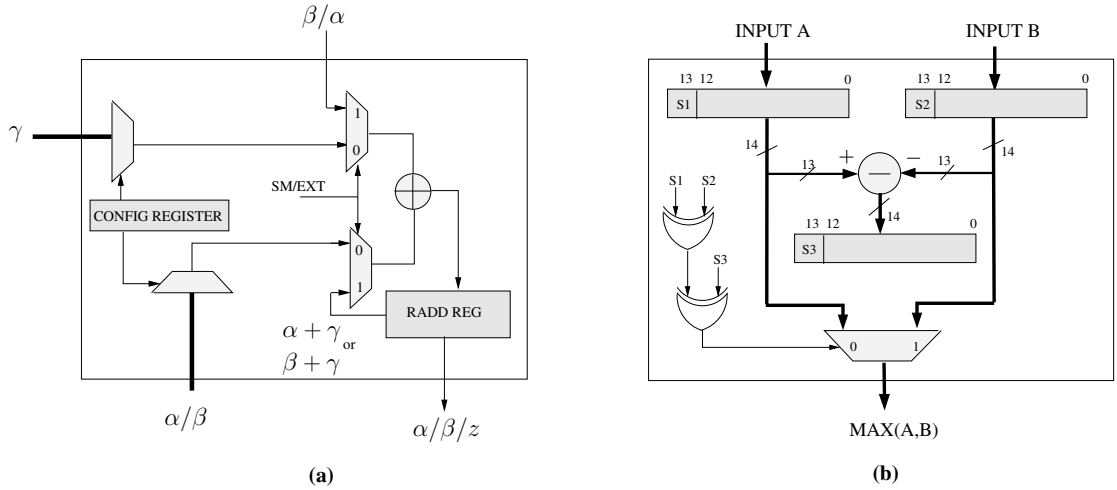


Figure 3.4 — Basic computational units of TurbASIP (a) Adder node (b) Modulo compare unit

Alpha (α), Beta (β) and Extrinsic Information (z) Computation: The issues related to compute α and β metrics are: (1) the successive addition of γ with α and β to compute them over all section of the trellis and (2) selecting the maximum of α and β related to the transitions associated with each state. Same case with extrinsic information where all three metrics α, β and γ are added on each section of trellis. But for extrinsic information generation, the maximum operation is performed on values related to those transitions which are occurred due to the input for which extrinsic is being computed. In literature this operation is often referred as Add Compare Select (ACS) operation.

The basic computational units of TurbASIP providing ACS are shown in Fig.3.4. An Adder Node (Fig.3.4(a)) can be used for addition operation required both in state metric and extrinsic information. While computing state metrics, the adder node can be configured for one of the input state metric (α of previous symbol or β of next symbol) and associated γ using configuration register, depending upon the trellis selected.

While performing addition operation involved in extrinsic information computation, the already stored sum (in RADD REG) of state metric and branch metric ($\alpha+\gamma$ or $\beta+\gamma$) is added with other state metric (β, α respectively). The quantization for the state metrics and extrinsic information is based on

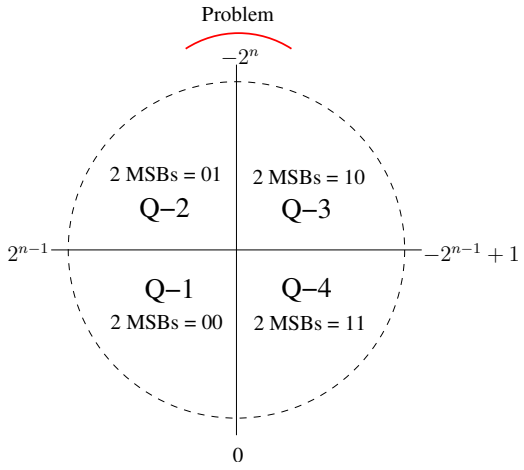


Figure 3.5 — Modulo algorithm extrinsic information processing

modulo algorithm [72] and the value are allowed to overflow. When overflow occurs, causing values to enter from positive region to negative region, the largest value becomes the smallest. In this situation when maximum finding operation is performed, a simple maximum operation can not be applied. To address this issue the architecture of specialized max. operator is shown in Fig.3.4(b) which detects the largest value even in case of overflow conditions and conforms to modulo algorithm.

With the considered modulo algorithm, particular scaling is required when the different extrinsic informations related to one symbol lay in second and third quadrant of the 2's complement number representation of Fig. 3.5. In fact in case of overflow, the largest extrinsic information moves from positive to negative region and becomes smallest which convey wrong information to other component decoder. Hence the first issue is to detect this particular situation which can be done by analyzing the 2 MSBs of generated extrinsic informations. As shown in Fig. 3.5, in which n bits represent the quantization of state metrics and extrinsic information, if an extrinsic information lies in Q-2 its two MSB's will be "01" whereas in Q-3 they will be "10". Hence if some of extrinsic informations related to different combinations of a symbol lay in Q-2 and others in Q-3 this will identify the problematic situation. In this case the second step is to correct the extrinsic information in a way that the largest extrinsic information remains largest. This can be done simply by an unsigned right shifting of all the extrinsic informations.

Forward Backward Recursion Units: As stated above, butterfly scheme of state metric and extrinsic information generation is used. Hence, two hardware recursion units, one working in forward and the other in backward direction are used. Since there are 32 transitions in a 16-state single binary or 8 state double binary code, one recursion unit is made up of 32 adder nodes. Hence, 64 adder nodes are used in TurbASIP. The arrangement of adder nodes in Forward recursion unit for double binary code

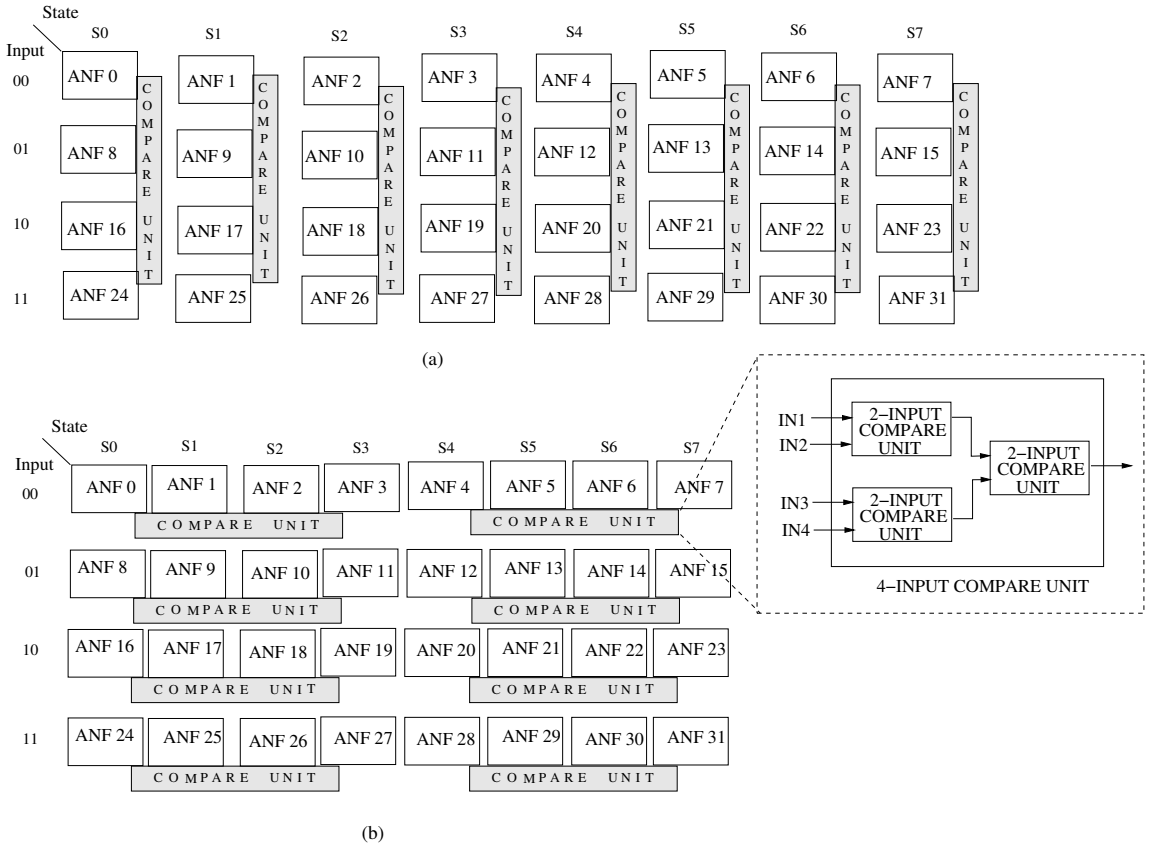


Figure 3.6 — Forward Recursion Unit composed of 32 ANF (Adder Node Forward) and 8 4-input Compare Unit (a) Compare Units used for state metric computation (b) Compare Units used for extrinsic information computation

is shown in Fig.3.6. The adder nodes in the rows receive proper γ due to 4 possible inputs in a double binary code whereas 4 adder nodes in each columns represent 4 state metrics related to each state. Same architecture is used for backward recursion unit. As stated above using configuration register each adder node receives its inputs according to the trellis structure.

To perform the max. operation, 24 2-input compare units are used in one recursion unit. These 24 2-input compare units can be configured to work as 8 4-input compare units. When computation of max. operation in state metrics generation is required, 8 4-input compare units are connected to the four outputs (output of 4 adder nodes in a column) of each column of recursion unit as shown in Fig.3.6(a). Hence, at the output of 8 four-input max. operators, there are 8 state metrics of double binary code. In case of extrinsic information computation, as shown in Fig.3.6(b), the 8 4-input compare units are connected to adder nodes in rows in such a way that 8 elements of a row are divided into 2 sets of 4 adder nodes (first set made up of 4 adder nodes on the left of the row and the second set made up of 4 adder nodes on the right). These 8 sets from 4 rows are connected to same 8 4-input compare units. Hence, with 8 4-input compare units user has two biggest candidates per row at the output of 4-input compare units. The two largest values in a row a saved back in the RADD registers of first two adder nodes of that row. Reusing 4 2-input compare units one can find the maximum between these two candidates which is the extrinsic information for each combination of input bits.

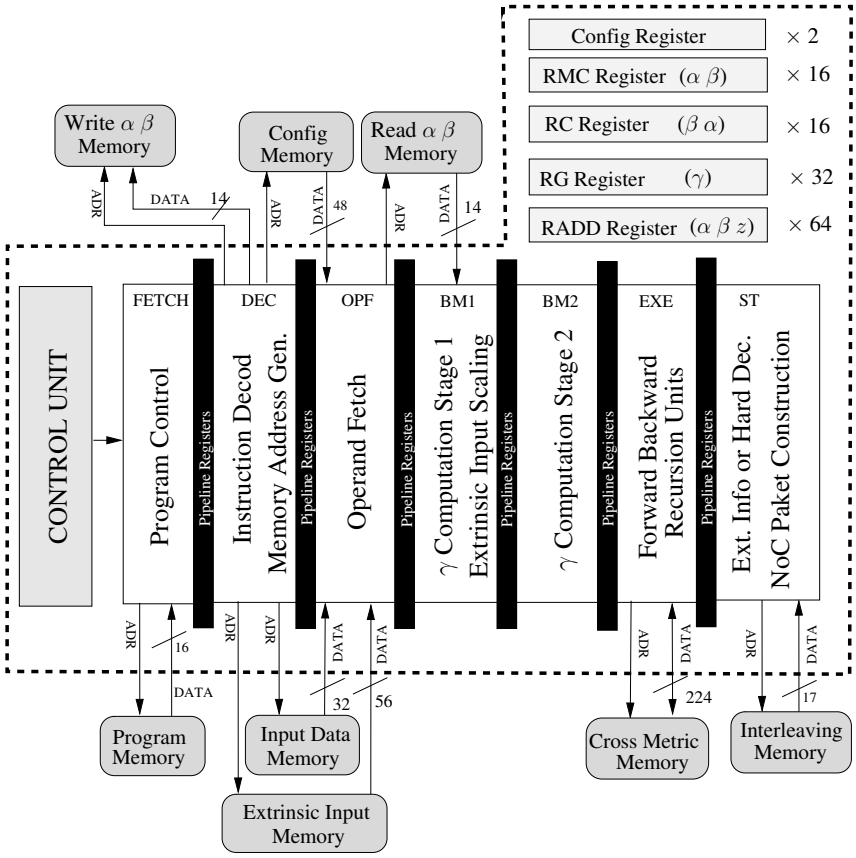


Figure 3.7 — TurbASIP architecture

3.4.1.2 Complete TurbASIP Architecture

TurbASIP architecture is composed of memory interface, internal registers, basic building blocks and a control unit as shown in Fig.3.7. Regarding memory interface, the application program is saved in the program memory. Config Memory is used to store different configuration of trellis which can be loaded in TurbASIP to switch between different trellis structures. The input data to the ASIP for data decoding is provided from Input and Extrinsic Data memories. To achieve butterfly scheme these two memories are further divided into top and bottom banks. Cross metric memory is used to store the state metrics while left side of butterfly scheme is in progress where as these stored metrics are used to compute the extrinsic information during right side of the butterfly scheme. The interleaving/deinterleaving tables are stored in the interleaving memories. Once TurbASIP computes the extrinsic information, the interleaving address from these memories are read. This address is placed as header whereas the extrinsic information is placed as payload in the packet which is sent on the network. Finally Read/Write α, β Memories are used to store the last values of state metrics. In the context of sub-blocking parallelism these saved state metrics are used to implement message passing method of metrics initialization as shown in Fig.3.3.

Certain register banks are used for ASIP configuration and storage of different parameters associated to max-log-MAP algorithm. Two configuration registers are dedicated for the configuration of two recursion units of TurbASIP. The configuration of the ASIP is downloaded from Config. Memory into these registers. The TurbASIP work in the same configuration unless the contents of these registers are changed. RMC registers store the state metric after max operations whereas RC registers

are used to read state metrics during the computation of right side of butterfly scheme. Branch metric γ are stored in RG register. RADD registers are part of ACS units as shown in Fig.3.4(a).

The function of control unit is to manage all the resources spread over seven pipelines stages. After instruction fetch and decode pipeline stages, the third pipeline stage is dedicated to fetch the operand from input memories. Two next stages, BM1 and BM2, are for γ computation. In execute (EXE) pipeline stage, resources of Add Compare Select (ACS) operations are placed. The last pipeline stage is to compute the extrinsic information and hard decisions.

3.4.1.3 Sample Program of TurbASIP

To give an explanation on how TurbASIP can be used for a decoding application, Listing 3.1 presents the piece of code written for dual binary code of Wimax standard for the first iteration. The text after (;) sign shows the comments.

Listing 3.1 — TurbASIP: assembly code for 8-state double binary turbo code for first iteration

```
1  ;loading configuration in config. registers
2          LD_CONFIG 0
3          LD_CONFIG 1
4          LD_CONFIG 2
5          LD_CONFIG 3
6  ;setting block length
7          SET_SIZE 48
8  ;scaling of extrinsic information
9          SET_SF 6
10 ;uniform start values for alpha/beta
11         SET_RMC UNIFORM, UNIFORM
12 ;zero overhead loop instruction
13         ZOLB _LW1,_LW1,_RW1
14 ;left butterfly alpha/beta +gamma
15         DATA LEFT WITHOUT_EXT ADD M
16 ;max for alpha/beta
17 _LW1:    MAX2 STATE METRIC NOTHING
18 ;right butterfly alpha/beta +gamma
19         DATA RIGHT WITHOUT_EXT ADD M
20 ;max for alpha/beta
21         MAX2 STATE METRIC NOTHING
22 ;left butterfly alpha+beta +gamma
23         NO WITHOUT_EXT ADD I
24 ;first max for extrinsic computation
25         MAX2 SYSTEMATIC NOTHING
26 ;second max for extrinsic computation
27 _RW1:    MAX1 SYS_PARITY EXT DECOD FB
28 ;message passing implementation
29         EXC_REC ALPHA_BETA0 0
30         EXC_REC ALPHA_BETA0 1
31         EXC_REC ALPHA_BETA0 2
32         EXC_REC ALPHA_BETA0 3
33         EXC_REC ALPHA_BETA0 4
34         EXC_REC ALPHA_BETA0 5
```

```
35          EXC_REC  ALPHA_BETA0  6
36          EXC_REC  ALPHA_BETA0  7
```

The program starts with LD_CONFIG instruction which configure the TurbASIP for the Wimax trellis. Using set size SET_SIZE instruction programmer can set the number of double binary symbol in a frame. SET_SF is the command to scale the input extrinsic e.g with target double binary application the value 6 corresponds to multiplying the input extrinsic with 0.7 before computing the branch metrics. SET_RMC UNIFORM instruction sets the RMC for zero which means all starting states are equiprobable. ZOLB is the zero overhead loop instruction. With this single instruction, the next two lines of code executes 24 times (half of frame size) which is in fact implementing left side of butterfly decoding scheme for 24 symbols without using the extrinsic information. The reason of not using the extrinsic information is due to the fact that during first shuffled iteration the extrinsic information memories hold the data of last iteration of previous frame. After this, the next five instruction (from "DATA RIGHT.." instruction to "MAX1 SYS_PARITY..") execute 24 times hence, implementing right side of the butterfly decoding scheme. During the execution of left butterfly, the first instruction "DATA LEFT.."is used to compute state metrics related to all transitions. The next "MAX2 STATE METRIC" perform compare operation to compute 8 state metrics both in forward and backward directions. In the right part of the butterfly scheme, the first two instructions compute the state metrics and next three instructions compute the extrinsic information. The first instruction to compute extrinsic is related to performing the summation of α, β and γ while next two instructions are used to do compare operation on the rows of recursion units in two steps. Finally at the end of processing of the block of 48 symbols, the ASIP initializes its state metric register (RMC) using message passing for next iteration. Hence, during one iteration, 2 clock cycles per 2 symbols are used in left side of butterfly decoding scheme and 5 clock cycles per 2 symbols are used in right side of the butterfly decoding scheme.

After the first iteration the extrinsic information memories hold the right extrinsic information hence one can use them. The code for next five iterations is shown in Listing 3.2 where instructions for DATA_LEFT and DATA_RIGHT with READ_EXT option in place of WITHOUT_EXT option of Listing 3.1.

Listing 3.2 — TurbASIP: assembly code for 8-state double binary turbo code for iteration number 2-6

```
1  REPEAT UNTIL _loop 5 times
2  ;zero overhead loop instruction
3           ZOLB _LW0,_LW0,_RW0
4  ;left butterfly alpha/beta +gamma
5           DATA LEFT READ_EXT ADD M
6  ;max for alpha/beta
7  _LW0:    MAX2 STATE METRIC NOTHING
8  ;right butterfly alpha/beta +gamma
9           DATA RIGHT READ_EXT ADD M
10 ;max for alpha/beta
11          MAX2 STATE METRIC NOTHING
12 ;left butterfly alpha+beta +gamma
13          NO READ_EXT ADD I
14 ;first max for extrinsic computation
15          MAX2 SYSTEMATIC NOTHING
16 ;second max for extrinsic computation
17 _RW0:    MAX1 SYS_PARITY EXT DECOD FB
18 ;message passing implementation
```

```
19              EXC_REC  ALPHA_BETA0  0
20              EXC_REC  ALPHA_BETA0  1
21              EXC_REC  ALPHA_BETA0  2
22              EXC_REC  ALPHA_BETA0  3
23              EXC_REC  ALPHA_BETA0  4
24              EXC_REC  ALPHA_BETA0  5
25              EXC_REC  ALPHA_BETA0  6
26              EXC_REC  ALPHA_BETA0  7
27  _loop       NOP
```

To compute the hard decision in last iteration, the assembly code is presented in Listing 3.3.

Listing 3.3 — TurbASIP: assembly code for 8-state double binary turbo code for last iteration

```
1  ;zero overhead loop instruction
2           ZOLB _LW2,_LW2,_RW2
3  ;left butterfly alpha/beta +gamma
4           DATA LEFT READ_EXT ADD M
5  ;max for alpha/beta
6  _LW2:    MAX2 STATE METRIC NOTHING
7  ;right butterfly alpha/beta +gamma
8           DATA RIGHT READ_EXT ADD M
9  ;max for alpha/beta
10          MAX2 STATE METRIC NOTHING
11 ;left butterfly alpha+beta +gamma
12          NO READ_EXT ADD I
13 ;first max for extrinsic computation
14          MAX2 SYSTEMATIC NOTHING
15 ;second max for extrinsic computation
16 _RW2:    MAX1 SYS_PARITY HARD FB
```

The hard decisions are computed by using HARD option in the last instruction of Listing 3.3 which was EXT DECOD in Listing 3.1 & 3.2.

As far as the throughput is concerned, if the overhead caused by other instructions is neglected, 3.5 clock cycles per symbol are required to generate extrinsic information per iteration (or hard decision during the last iteration).

3.4.2 NoC Based on Butterfly Topology

The Butterfly network is a multistage interconnection network with 2-input 2-output routers and unidirectional links. The advantages of this topology are: first, the logarithmic diameter of the network ($log_2 P$ with P the number of network input ports) which gives a number of routers equal to $\frac{P}{2} log_2 P$; secondly the recursive structure of the network (a network of diameter d is obtained with two networks of diameter $d-1$) which enables high scalability; and finally a very simple routing scheme that uses directly the bits of the destination address for the selection of the output port at each stage of the network. However, this type of network does not have path diversity: there exists only one route between each source and each destination, which increases the risk of conflicts in the routers. To mitigate this problem, queues to store the conflicting packets are used. Several architectural decisions are made in the proposed Butterfly network according to the specificities of the supported application.

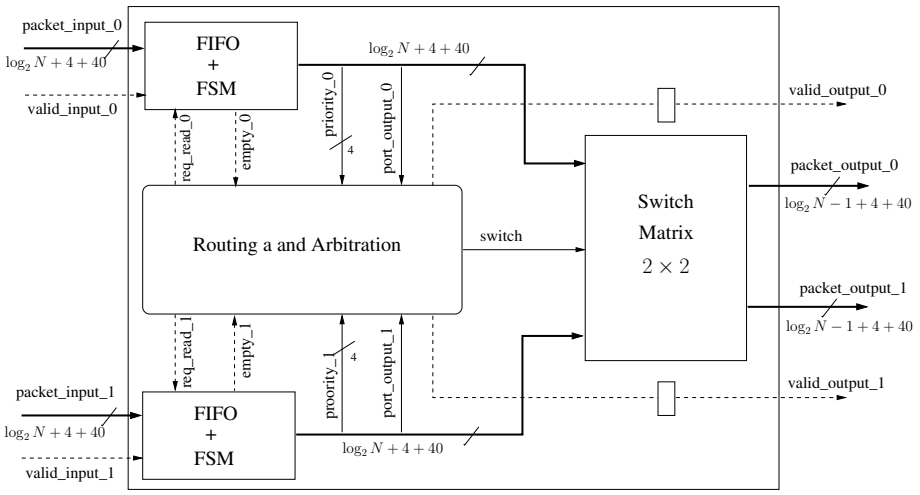


Figure 3.8 — Router architecture of Butterfly based NoC

First of all, in the considered turbo decoder architecture, the interleaving (respectively deinterleaving) function of extrinsic information must be supported by the interconnection network. For this purpose, it is necessary that the network can vehicle any permutation of the inputs to its outputs, which is the case for the Butterfly network. Thus, through the packet addressing, the interleaving (respectively deinterleaving) of the data is performed due to the identifier of destination port and the destination memory write address containing extrinsic information. With TurbASIP implementing the butterfly decoding scheme, a maximum of two packets will be generated by each ASIP at the inputs of the network. This is why the number of input ports of the network is twice larger than the number of TurbASIP. Fig.3.3 represents a bidirectional Butterfly network (made up of two unidirectional Butterfly networks) that connects 4 TurbASIPs each producing 2 packets for 8 memories attached to destination TurbASIP. Network Interfaces (NI) are used between network and processor /memories. The Destination-Tag routing technique is used. This deterministic routing uses a digit of the destination address in the header of the packets to select the output port at each router along the path from the source to the destination.

Besides the payload which is composed of the extrinsic information, the packets contain a simple header field. The header includes the previously defined routing information and the destination memory address. The first field has a width of d bits, d being the diameter of the network and the second field having a width of $\lfloor \log_2(\frac{N}{P}) \rfloor + 1$ bits, where N is the length of the frame to be decoded and P the number of TurbASIPs in an interleaving domain. Router architecture (Fig.3.8) implements a simple router with 2 input and 2 output ports with input FIFOs to store the conflicting packets. FIFO depth is determined with respect to the worst case which is the case when all input packets have the same network output port. Thus, FIFO depth is fixed to $2i$ for all routers of stage i of the Butterfly network (i varies from 0 to $log_2(P)$). The Routing and Arbitration block implements, in the considered version, a round-robin queues serving policy. It generates the control signals for the switch matrix, FIFOs, and output packets.

3.5 Towards Heterogeneous Multi-ASIP and NoC Based Flexible Turbo Receiver

Based on the promising results obtained for the high throughput flexible turbo decoder and for the parallel turbo demodulation/equalization study, presented in chapter 2, Fig.3.9 proposes an heterogeneous multi-ASIP NoC based architecture for future high throughput flexible radio platform.

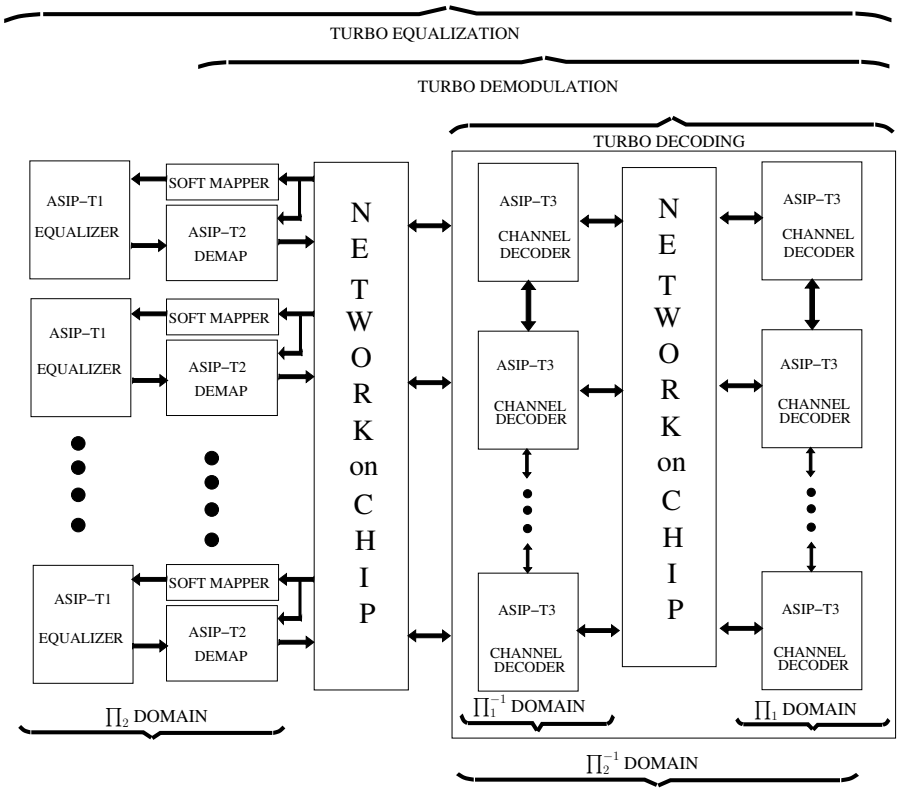


Figure 3.9 — Heterogeneous multi-ASIP and NoC architecture for turbo receiver

To achieve turbo decoding, turbo demodulation and turbo equalization, three different types of ASIPs are required. The first ASIP of type T1 is for equalization which should be efficiently flexible for the channel characteristics mentioned in Chapter 1 and the MIMO STC codes used on transmission side. T2 type of ASIP is for demapping and should be reusable both in single antenna and MIMO transmissions. It should be flexible for different constellation characteristics (size, mapping and rotation) and should have the capability to use the *a priori* information in iterative demodulation context. T3 ASIP is for max-log-MAP decoding providing flexibility for the target trellises. The soft mapping function is only used in turbo equalization for MIMO and does not have many flexibility parameters. Hence, some simple parameterized circuit can serve the purpose. The integration of these ASIPs is done in such a way that the NoCs are placed between those ASIPs which are supposed to receive interleaved/deinterleaved data during turbo processing.

In this platform requirements and their proposed solution are mapped on hardware components in following way:

- Flexibility parameters for a processing block and efficient metric generation level parallelism is implemented inside the ASIP.

- Component level parallelism is achieved through the use of multiple instances of heterogeneous ASIPs.
- Solution to memory access conflicts, caused by the component level parallelism and interleaving rules, and shuffled turbo processing is mapped on NoCs.
- Utilization of desired processing power for a specific system configuration can be achieved by switching on/off required number of ASIPs in the architecture and hence providing a simple way to achieve an energy-efficient platform.

To conceive a turbo receiver having qualities of offering flexibility and scalability in processing power, proposed in Fig. 3.9, the work in this thesis is focused on implementing other ASIPs for equalization and demapping tasks. The other issue is to modify the TurbASIP to generate the extrinsic and *a posteriori* information for the demapper and soft mapper. The final task is to integrate all the processing units through NoCs. The next three chapters are dedicated individually for equalizer ASIP, Demapper ASIP and prototyping of a flexible and scalable turbo receiver.

3.6 Conclusion

In this chapter an effort is made to present an architecture conforming to the requirement laid down in chapter 1 which are: flexibility for transmission parameters, error rate performance approaching theoretical limit and use of required processing power of the system for specific system configurations. To address these requirements we discussed them gradually and proposed their possible hardware solution. Taking the example of turbo processing which manages error rate performance at the cost of high latency and low throughput. The solution to the problem of latency and low throughput, associated with iterative processing, is addressed by studying the parallelism in chapter 2. The first issue is the best use of metric generation level parallelism while retaining the flexibility of a processing unit. For this ASIP methodology is discussed and multi ASIP architecture is presented to implement sub-blocking. Although sub blocking parallelism provides gain in throughput but it gives birth to the communication conflicts among the processing units due to the presence of interleavers. Use of NoC is an efficient method to counter the communication conflicts. For design approach illustration, a multi-ASIP NoC based architecture is presented as a case study for the realization of a turbo decoder is also presented. Finally a global heterogeneous multi-ASIP NoC based architecture is presented for a high throughput flexible turbo receiver.

CHAPTER

4 EquASIP: ASIP-based MMSE-IC Linear Equalizer

THIS chapter presents the ASIP architecture dedicated for MMSE-IC linear equalization algorithm, namely EquASIP. The presented ASIP exploits the first level of parallelism available in equalization application, introduced in chapter 2 as Symbol Estimation Level Parallelism. Proposed EquASIP architecture can be used for multiple MIMO space time codes both in an iterative and a non-iterative context and provides support for Alamouti Code, 2×2 Golden code, and 2×2, 3×3 and 4×4 spatially multiplexed MIMO-OFDM environment using BPSK, QPSK, 16-QAM and 64-QAM modulation schemes. Furthermore, EquASIP is designed to be modular in order to facilitate its integration in a scalable multiprocessor platform (exploiting the second parallelism level).

This chapter is organized in the following order. First of all, a brief state of the art section is provided to summarize the available hardware implementations related to this domain. Flexibility parameters are then analyzed to make a choice about the hardware resource allocation and sharing. The third section presents in a bottom-up approach the detailed architecture of the required complex number operators. Based on these operators, the complete EquASIP architecture together with the proposed instruction set and sample programs are presented in the subsequent three sections. Finally, the last section provides synthesis results, execution performance and comparison with state of the art implementations.

4.1 State of the Art

State of the art MIMO detection techniques can be classified in three categories [73]: ML detection, Sphere Decoding (SD) and linear filtering based detection. The complexity of ML detection increases exponentially with the number of antennas and modulation order. The SD approach has a polynomial complexity. To perform SD, first a QR decomposition of channel matrix is carried out and then tree exploration is performed. This tree search is further categorized as depth-first and breadth-first methods. The depth-first has a reduced area complexity and optimal performance, but has variable throughput with SNR. In breath-first case, the most famous algorithm is the K-best in which K best nodes are visited at each level. Hence, the complexity depends on K. A large value of K results in high complexity and good performance. Linear filtering based solutions such as MMSE-IC, considerably reduce the complexity of the hardware implementation of a MIMO detector. Whereas the compensation for sub-optimality can be achieved using turbo equalization.

In linear filtering based solution, matrix inversion implying complex numbered operations is the most demanding computational task. Hence, most of the existing work has been focused on the inversion of variable-sized complex-numbered matrices. Matrix inversion based on QR Decomposition Recursive Least Square (QRD-RLS) algorithm has been proposed [74]. In [75], authors have proposed a Coordinate Rotation Digital Computer (CORDIC) and Squared Givens Rotation (SGR) based Linear MMSE detector while in [76] a linear array architecture for SGR implementation has been introduced. Matrix inversion through block-wise analytical method has been implemented in [77]. Two separate MMSE-IC2 equalizers for 4×4 turbo MIMO SM environment using QPSK and QAM-16 modulations, implementing CORDIC method of QR decomposition, have been proposed in [78] for fast fading applications. Using analytic method of matrix inversion, a fully dedicated architecture for MMSE-IC1 LE for 2×2 turbo MIMO system with pre-coding used in quasi static channel has been proposed in [79]. The other work carried out in [80] shows exciting results in terms of throughput for 802.11n MIMO-OFDM application. The implementation is based on a inverse free architecture using square-root MMSE formulation.

To the best of our knowledge all the available implementations target a specific STC with limited modulation support. In the following sections, the process of developing a flexible MMSE-IC equalizer using the ASIP approach and conforming to multi-standard requirements is explained. Performance comparison of the proposed ASIP with above referenced state of the art works is provided at the end of this chapter.

4.2 Flexibility Parameters and Architectural Choices

The flexibility parameters influencing the equalizer architecture, based on MMSE-IC equalization algorithm, comes from different sources. Depending upon these parameters, architectural choices can be made for efficient hardware resource allocation and sharing.

4.2.1 Flexibility Parameters

Following are the three considered sources in extracting the flexibility parameters:

- MIMO STC supported at the transmitter
- Time diversity of the channel
- Possibility of iterative equalization in the receiver

MIMO STC: MIMO Spatial multiplexing (SM), Alamouti code and Golden code are the STCs adopted in emerging wireless standards. For MIMO SM with different antenna dimensions, such as 2×2, 3×3 and 4×4, the expressions (2.41 to 2.47) can directly be implemented using channel matrix and received vector inputs. Hence, a hardware capable of implementing variable sized complex matrix operation involved in the algorithm can address MIMO SM from 2 to 4 antennas. As far as Golden code and Alamouti code are concerned, MMSE-IC algorithm can be used by applying equivalent channel transformations on the inputs prior to their use. In case of 2×2 Golden code, the equivalent channel transformation is presented in [81]. The idea is to treat two transmitted vectors (each having two elements) as one transmission of four symbols. By applying equivalent channel transformation, the inputs to the MMSE-IC equalizer are $\mathbf{y}$ of four elements and an equivalent channel matrix $\breve{\mathbf{H}}$ of size 4×4. The equivalent channel transformation of Alamouti code is presented in [82] which transforms a 2×1 channel matrix into a 2×2 equivalent matrix and 2×2 channel matrix into a 4×4 equivalent matrix. Hence, supporting MIMO SM with an additional capability of equivalent channel transformation, addresses this first source of flexibility parameters.

Time Diversity: The time diversity of the channel decides how frequent the computations of equalization coefficients (2.41, 2.44 and 2.46) is required. For quasi static channel these coefficients are computed once per iteration whereas for fast fading channel they are computed for each received vector per iteration. In case of block fading, these coefficients are computed for a set of received vectors for which channel matrix is considered as constant.

Iterative Equalization: The last source of flexibility is the iterative/non-iterative nature of the equalizer. In an iterative context the equalizer must incorporate the *a priori* information.

4.2.2 Architectural Choices

In the MMSE-IC algorithm, presented in subsection 2.3.1, one can note that the expressions computing equalization coefficients and symbol estimation exhibit similar arithmetic operations. Now considering the flexibility need related to time diversity of the channel, allocating separate resources for equalization coefficients computation will result in an inefficient architecture in case of quasi static and block fading channel. For this reason, and targeting flexibility as well as efficiency, our first architectural choice is based on hardware resource sharing between these two tasks.

Out of these two distinctive parts of the algorithm, the one related to equalization coefficient computation is more resource demanding. In fact, in this part of the algorithm, the implied computations can only be done in a serial order. For example, to compute matrix $\mathcal{F}$ (2.44), one need to compute:

- Hermitian of $\mathbf{H}$
- Matrix multiplication $\mathbf{H}\mathbf{H}^{\mathbf{H}}$
- Scaler-Matrix multiplication
- Matrix addition
- Matrix inversion

The other metrics (such as β_k, λ_k and g_k) are computed with a similar pattern. For this kind of serial computations, temporal parallelism implementation through pipelining can be applied to increase

throughput. Now considering the flexibility need related to STC, allocating hardware resources according to the requirements of the most complex STC configuration will result in an inefficient architecture for the low complexity configurations. For this reason, our second architectural choice is based on dimensioning the hardware resources in order to be fully used in all STC configurations. In this regard, the implied complex matrix operations are analyzed and broken down into basic arithmetic operations. Then adequate hardware operators are constructed considering the best tradeoff between flexibility, parallelism and hardware efficiency.

4.3 Hardware Architecture for Basic Operators

In this section, a bottom-up presentation approach is adopted to explain the proposed hardware architecture capable of performing complex operations through the basic arithmetic operators.

4.3.1 Complex Number Operations

In MMSE-IC algorithms, the complex matrix operations can be broken down into basic complex number operation such as addition, subtraction, negation, conjugation and inversion. To perform each operation the architecture of the operator is detailed below.

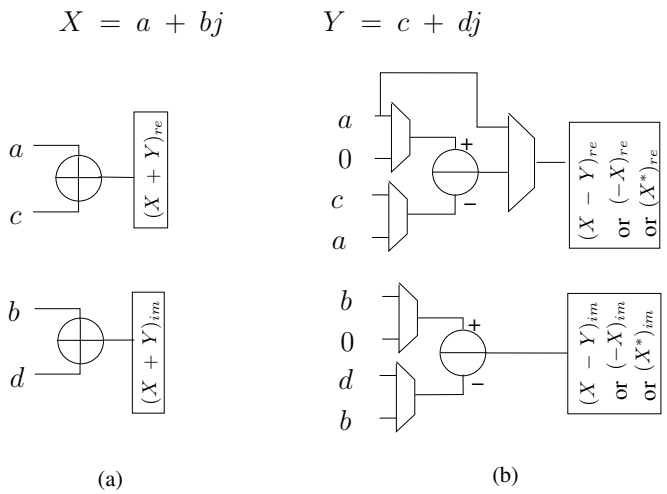


Figure 4.1 — Basic components (a) Complex adder (b) Complex subtracter, negater and conjugator

4.3.1.1 Complex Number Addition, Subtraction, Negation and Conjugate

The complex number addition needs two real adders whereas a complex numbered subtraction needs two real subtracters. Using two real subtracters, negation of a complex number can be performed. Similarly, conjugate of a complex number, required in calculating the hermitian of a matrix can also share the real subtracter. Fig.4.1(a) shows hardware architecture for addition of two complex numbers $X = a + jb$ and $Y = c + jd$ whereas Fig.4.1(b) shows combined architecture of subtraction of X and Y and negation/hermitian of a complex number X.

4.3.1.2 Complex Number Multiplication

By applying the classical formula (4.1) of multiplication of complex numbers, a complex numbers multiplier must perform 4 real multiplications and 2 real additions/subtractions.

$$X \times Y = (a + jb)(c + jd) = (ac - bd) + j(ad + bc) \tag{4.1}$$

A rearrangement may be proposed to reduce the number of multiplications required, as:

$$X \times Y = (a + jb)(c + jd) = a(c + d) - d(a + b) + j\left[a(c + d) + c(b - a)\right] \tag{4.2}$$

By applying this reformulation, a complex number multiplier must perform only three real multiplications and 5 real additions/subtractions. Reducing one real multiplier per complex multiplier at the cost of three adders significantly reduces the complexity of the complex number multiplier. In addition the adders and subtracters of first stage of pipelined multipliers can also be used for complex number addition, subtraction, negation and conjugation. A Combined Complex Adder Subtracter and Multiplier (CCASM) is shown in Fig.4.2. This architecture is capable of performing all basic operation of complex number addition, subtraction, negation, conjugation (output at first stage of pipeline) and multiplication (output at third stage of pipeline).

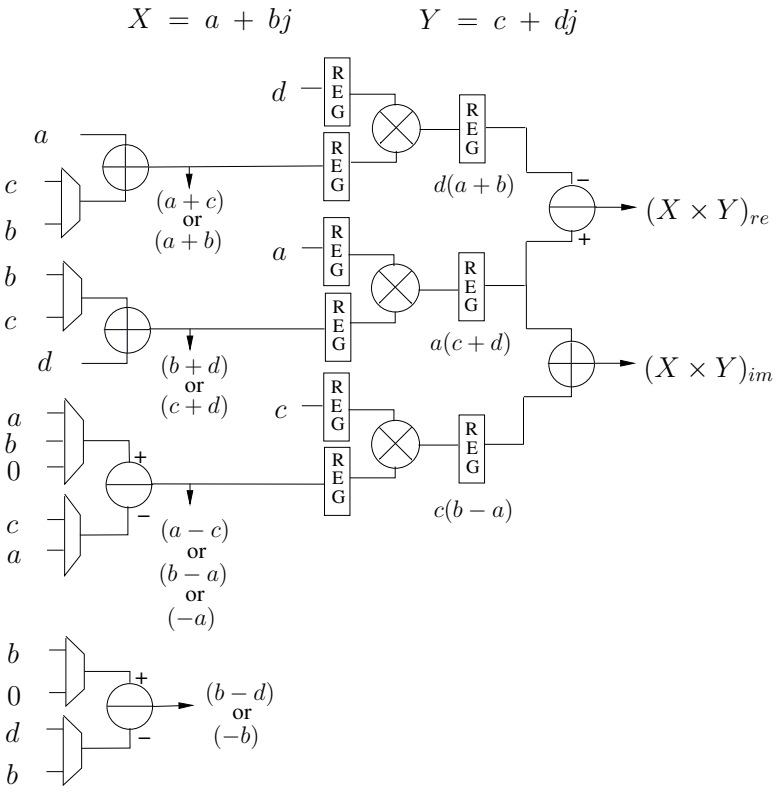


Figure 4.2 — Combined Complex Adder Subtracter and Multiplier (CCASM)

4.3.1.3 Complex Number Inversion

The inverse of a complex number can be computed using following expression:

$$\frac{1}{a + bj} = \frac{a}{a^2 + b^2} - \frac{b}{a^2 + b^2}j \tag{4.3}$$

The architecture for this inverter can be obtained by reusing the real multipliers and one adder of the CCASM to compute $a^2 + b^2$. Pre-computed LUT can then be used to find inversion value of $\frac{1}{a^2+b^2}$. Finally, two real multipliers and one subtracter are required for final result computation.

4.3.2 Complex Matrix Operations

In this subsection we propose the use of basic operators, developed in previous part, to achieve complex numbered matrix operations such as matrix hermitian, multiplication and inversion.

4.3.2.1 Matrix Hermitian, Addition, Subtraction, Negation

To perform hermitian operation on a matrix, at first, one need to copy the rows of the matrix into columns of an intermediate matrix. Then by taking complex conjugate of each element of this intermediate matrix, the resultant matrix will be the required hermitian matrix. Using 4 instances of the architecture presented in Fig.4.1 with some control logic, provides a fully parallel and flexible architecture to perform Matrix Hermitian, Addition, Subtraction and Negation operations for 2×2 and 4×4 matrices. In case of 3×3 matrix this architecture will be 75% efficient. Hence, to perform any of these operation on 2×2, 3×3 and 4×4 matrices, 1, 3 and 4 clock cycles will be required.

4.3.2.2 Matrix Multiplication

To perform a multiplication of two 2×2 matrices, 8 complex multiplications are required whereas for 3×3 and 4×4 matrices the number of complex multiplications required are 27 and 64 respectively. Use of four CCASM (Fig.4.2), can efficiently perform all operations (matrix hermitian, addition, subtraction, negation and multiplication) required for 2×2 and 4×4 matrices. For 2×2 matrix multiplications, two complex adders will be required to sum up the multiplication results whereas in 4×4 case, in addition to two complex adders, one more adder will be required. The architecture of 2×2 and 4×4 matrix multiplications is shown in Fig.4.3. The number of cycles required to perform 2×2, 3×3 and 4×4 matrix multiplications will be 2, 9 and 16 respectively.

4.3.2.3 Matrix Inversion

The matrix inversion can be achieved through one of the following methods:

- based on matrix triangulation
- based on analytical method

The first method based on matrix triangulation can realized using systolic architecture through the LU decomposition, Cholesky decomposition or QR decomposition. The method based on QR decomposition is the most interesting due to its numerical stability and its practical feasibility. It consists of decomposing decompose a matrix $\mathbf{A}$ of size $N \times N$ as $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is an orthogonal matrix ($\mathbf{QQ}^H = \mathbf{I}$) and $\mathbf{R}$ an upper triangular matrix. This decomposition allows to compute the inverse of the matrix $\mathbf{A}$ after a simple inversion of the triangular matrix $\mathbf{R}$ and a matrix multiplication as $\mathbf{A}^{-1} = \mathbf{R}^{-1}\mathbf{Q}$. There are several methods [83] to achieve this decomposition, such as the Givens method or the method of Gram-Schmidt. Hardware designers give special attention to the Givens method due to its practical feasibility, its parallelism and its numerical stability [84][76]. The method of Givens consists of triangularization of matrix $\mathbf{A}$ by applying a series of plane rotations

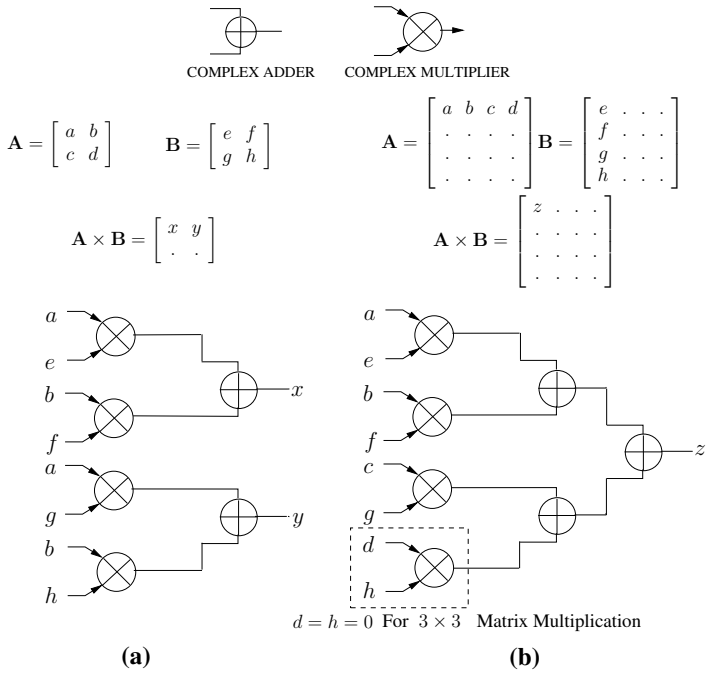


Figure 4.3 — Complex matrix multiplications (a) 2×2 Matrix multiplication (b) 3×3 and 4×4 Matrix multiplication

called Givens rotations. Each rotation is designed to cancel an element of $\mathbf{A}$. The standard method of Givens uses operations that are not easily implementable, including square root and division. Therefore, there are several variants of this method to avoid these operations. The SGR (Squared Givens Rotations) [85] and CORDIC method [86] are the best known methods. A comparison between the two approaches: SGR and CORDIC has been made by Myllyla $et\ al.$ [84] through MMSE detector. The results show that the CORDIC-based architecture is more expensive in hardware cost and is 1.5 times slower than those based on SGR. In his thesis work , Edman [87] used SGR method to achieve matrix inversion and studied both triangular and linear architectures. For this type of architecture there are dedicated Processing Elements (PEs) which are used as boundary elements and internal elements of a systolic array or linear array [76]. Although linear array architecture is flexible for variable sized matrix inversion, it is dedicated to matrix inversion only.

The analytic method of matrix inversion is good candidate, not only for variable sized matrix inversion but also for resource reuse for other matrix computations. The expression for the inversion of 2×2 matrix through analytical method is given by:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{4.4}$$

To implement (4.4) the resources required are a complex number negater and a complex divider. For a 4×4 matrix, the matrix is divided into four 2×2 matrix and inversion can be achieved block wise.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} W & X \\ Y & Z \end{bmatrix} \tag{4.5}$$

where

$$\begin{aligned} W &= A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} \\ X &= -A^{-1}B(D - CA^{-1}B)^{-1} \\ Y &= -(D - CA^{-1}B)^{-1}CA^{-1} \\ Z &= (D - CA^{-1}B)^{-1} \end{aligned}$$

The inversion of a 3×3 matrix is performed by extending it to a 4×4 matrix. This can be done by copying all three rows of 3×3 matrix into first three rows of 4×4 matrix and then putting zeros in all elements of fourth row and fourth column where a 1 should be put on the intersection of fourth row and fourth column. The inversion can then be performed using the method mentioned above. The final result lies in first three elements of first three rows (or column). All the expressions involved in the inversion of up to 4×4 matrix can be achieved through already described matrix operations and will be used in the EquASIP.

4.3.2.4 Operator Reuse in Fixed-Point Representation

To find the required data width for fixed-point representation of the parameters involved in MMSE-IC algorithm, long simulations have been conducted for all supported system configurations (STC and modulation type). Results analysis have shown that at maximum 16-bit signed representation with different bits for integer and fractional part is sufficient for all the parameters involved during the different computational steps of MMSE-IC LE algorithm to ensure a performance loss below 0.2dB. To enable the reuse of hardware resources for these different computations, involving operands with different fixed-point representations, certain rules have been set. First of all, while reading input data from memories, the data which is represented in less than 16-bits, is sign extended to 16-bit. Secondly, a programmable 33 to 16-bit conversion is performed at the outputs of the multipliers. Last of all, to avoid the hazards caused by overflow/underflow during an arithmetic operation, a control mechanism is provided to fix the output at its maximum/minimum limit. Fig. 4.4 shows the 16-bit quantization in signed 2's complement representation of the different implied parameters. In Fig. 4.4, notation (x dot y) designates a singed number where x represents the number of bits for integer part and y represents the number of bits for fractional part. Furthermore, Fig.4.5(a) shows the FER performance of 2×2 MIMO SM with QPSK and Fig.4.5(b) shows the FER performance of 4×4 MIMO SM with 64-QAM.

4.4 EquASIP Architecture

The proposed ASIP architecture is mainly composed of Matrix Register Banks (MRB), Complex Arithmetic Unit (CAU) and Control Unit (CU) besides its memory interfaces. The input to the EquASIP are through "Channel Data Memory" and the soft mapper as shown in Fig. 4.6. The data bus of all inputs is set to 16 (32 bit for complex number). This provides flexibility to use up to 16 bit data representation and in case of smaller data widths, signed/unsigned extension can be done externally. The ASIP has 7 pipeline stages named as: FETCH, AD_SU_MUL1, MUL2, MUL3, 2ADD, 1ADD and OUT.

4.4.1 Matrix Register Banks

To store a complex number two separate 16-bit registers have been used, one storing the real and the other imaginary part. Based on the requirements of the expression (2.40) for a 4×4 spatially

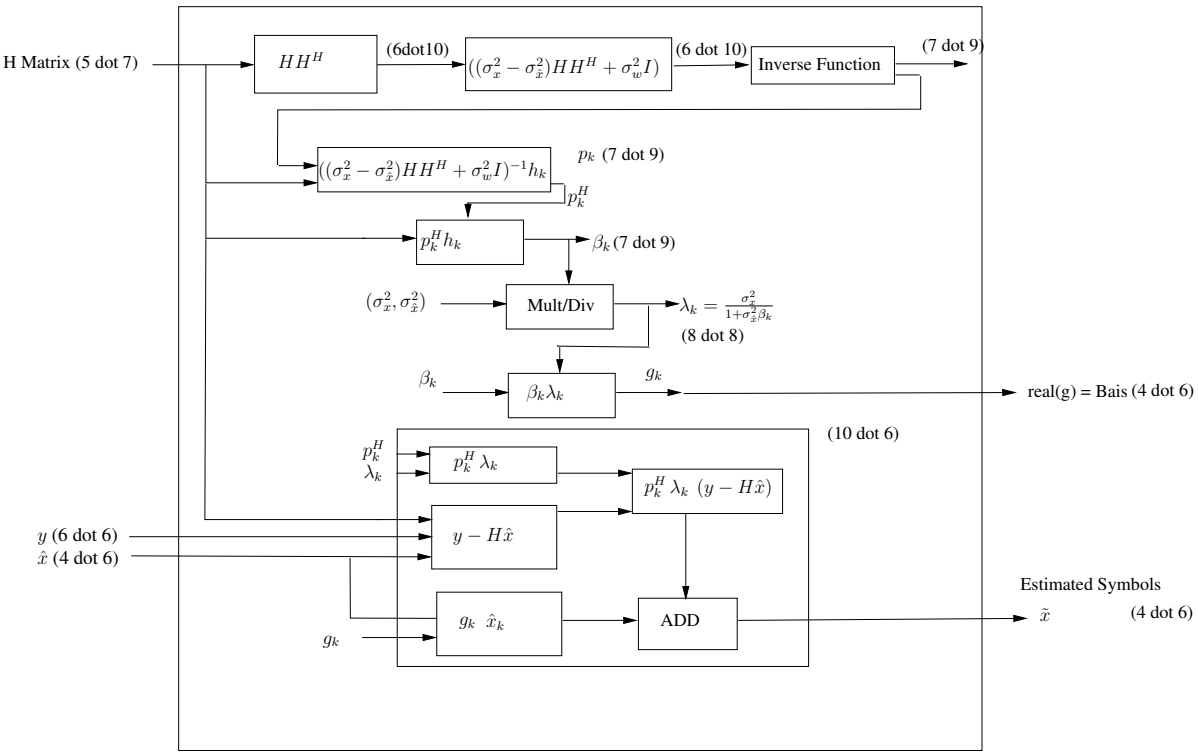
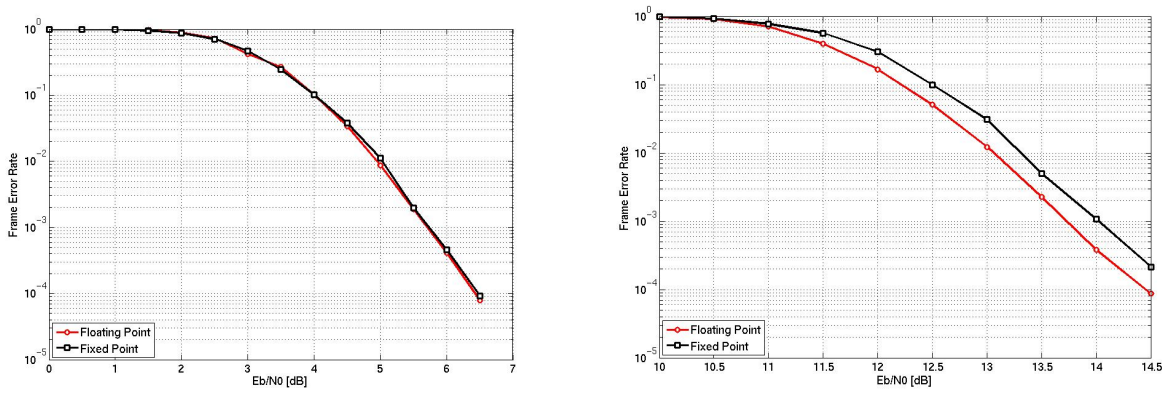


Figure 4.4 — Quantization Parameters for Fixed-point Representation



(a) (b)

Figure 4.5 — Floating point vs Fixed-point turbo equalization for 120 Source Byte, double binary encoder , $\frac{1}{2}$ code rate, Rayleigh fading channel (a) 2×2 QPSK ; (b) 4×4 64-QAM

multiplexed MIMO system, 13 MRBs have been proposed, where each MRB can store 4 complex numbers (Fig. 4.6). H-MRB (H0, H1, H2, and H3) which are connected to the memory, can store 4 rows or columns of Channel Matrix. Four V-MRB (V0, V1, V2, and V3) store 16 entries of $\lambda_k p_k$. GP0, GP1, GP2, GP3 and GP4 are assigned to the storage of g_j , $\hat{x}_j$, y, $g_j\hat{x}_j$ and the estimated symbols $\tilde{x}$ respectively. Other than this specific use, these GP registers save the intermediate results of equalization coefficients. Among other registers there are three registers to store the variances of noise, modulation symbol and decoded symbols besides pipeline registers and the registers for

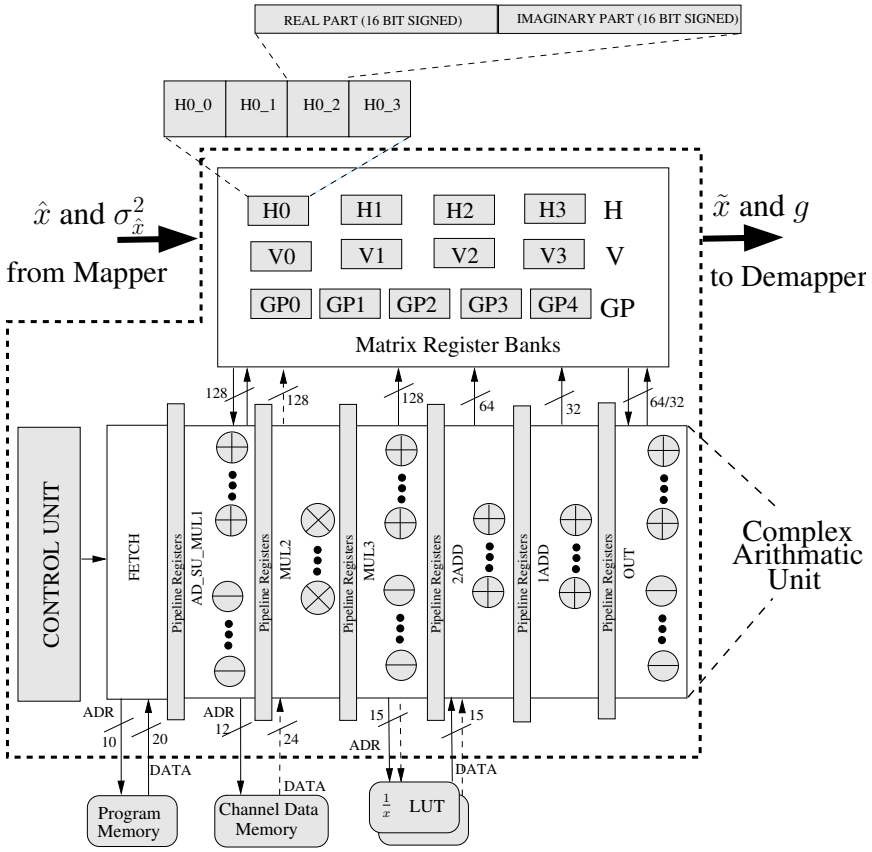


Figure 4.6 — EquASIP block diagram

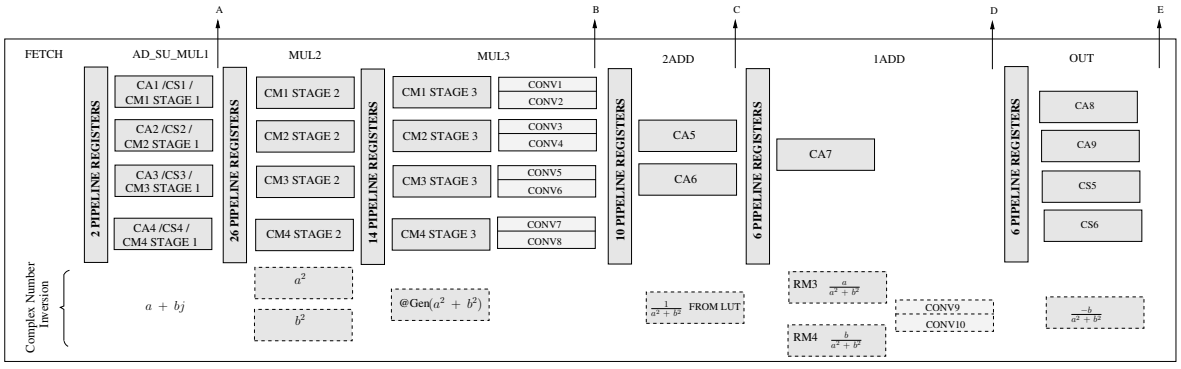


Figure 4.7 — CAU and pipeline stages

REPEAT instruction.

4.4.2 Complex Arithmetic Unit

The computational resources of the Complex Arithmetic Unit (CAU) of EquASIP are shown in Fig. 4.7. After fetch pipeline stage, 4 CCASM units (Fig. 4.2) are spread over three pipeline stages to perform 4 concurrent complex additions, complex subtractions/negation, complex conjugation and complex multiplications. The results of complex addition, subtraction, negation and conjugate operations are copied into destination registers in AD_SU_MUL1 pipeline stage. In MUL3 stage, 33-bit to 16-bit transformation is performed according to the information provided in multiply instruction. The results of four complex multiplication (16-bits for each of real and imaginary part of the complex number) are saved in the target registers. To perform 2×2 matrix multiplication one row/column of first matrix is introduced twice at first input of CCASMs and two columns/rows of second matrix are exposed to the second input of CCASMs. Providing the results of four complex multiplication to two complex adders in 2ADD pipeline stage, the output will give one resultant row/column of multiplication of 2×2 matrix. In case of 4×4 matrix multiplication, one row/column from each matrix goes to the inputs of four CCASM. The results of four multiplications are added together using 2 adders of 2ADD and one adder of 1ADD pipeline stage to output one element of 4×4 matrix multiplication. Complex adders/subtracters in last pipeline stage are used in the computation of Eq. (2.40). The inversion process of a complex number in different pipeline stages is shown as dotted area in Fig. 4.7. For this particular operation, additional resources are required as Look-Up Tables (LUT), two 33 to 16-bit converters, and two real multipliers.

4.4.3 Control Unit

The EquASIP control unit works as administrator of the 7-stage pipelined CAU as mentioned above and shown in (Fig. 4.6). It controls the flow of the program instructions over the designed datapath (MRBs, CAU) during the different stages of the pipeline. The functioning of the control unit will be reflected during the instruction set presentation which is detailed in the next section.

4.5 EquASIP Instruction Set

The instructions of the proposed ASIP are categorized as follows:

4.5.1 LOAD, MOVE, REPEAT, NOP

LOAD instruction is used to load channel matrix into H-MRB from memory. While loading data there are possibilities for loading directly or loading after applying conjugation to support equivalent channel transformation. LOAD_CODE instruction is used to initialize the V-MRBs for values which are used in equivalent channel transformation for Golden code. The MOVE instruction is used to transfer data between MRBs whereas REPEAT instruction repeats a block of code as many times as given in REPEAT_SIZE Register. NOP instruction is used to add empty cycles during the execution of the program when required.

4.5.2 Matrix Addition, Subtraction, Negation and Conjugation Instructions

The instruction format for addition, subtraction, negation and conjugation operation is shown in Fig.4.8. Besides opcode, the other fields are the “OPERATION” field and two “SOURCE” fields to input two register banks in complex adders and subtracters. The “OPERATION” field of 3-bits indicates

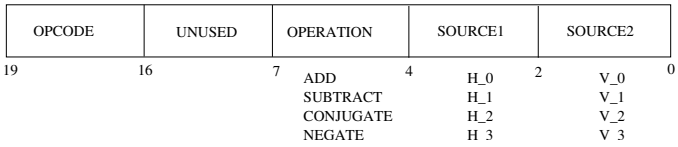


Figure 4.8 — 20-bit Addition, subtraction, negation and conjugate instructions

the following six different operations: ADD, SUBTRACT, CONJUGATE, NEGATE, MOV_REC and MOV_MOD.
ADD: Using ADD instruction, programmer can select any of the H-MRB as source1 and any of V-MRB as source2. The result of an addition is always saved in GP_0 MRB.
SUBTRACT: Using SUBTRACT instruction, any one of selected H-MRB and any of V-MRB are subtracted and result is always saved in GP_0 MRB.
CONJUGATE/NEGATE: In this single source instruction all four elements of one of the selected H-MRB are conjugated/negated and the results are copied in respective V-MRB i.e V-MRB(n) = Conjugate/Negate(H-MRB(n)) where n can be any integer from 0 to 3.
MOV_REV: This instruction copies the elements of H-MRB(0), in reverse order, into V-MRB(0) with second element in negative form. This is used to align the elements of 2×2 matrix (to be inverted) for a multiplication which results in its determinant. For example if H-MRB(0) has a matrix A with elements a, b, c and d (4.4) then V-MRB(0) will have elements $d, -c, b$ and a after the execution of this instruction. To obtain determinant of A ($det(A) = ad - bc$), one can multiply H-MRB(0) with V-MRB(0) and add the results of first two complex multiplications.
MOV_MOD: This instruction is to copy and rearrange the matrix A (saved in H-MRB(0)) in V-MRB(0) to a form required in the inversion of a 2×2 matrix (4.4) i.e. if H-MRB(0) has a matrix A with elements a, b, c and d then V-MRB(0) will have elements $d, -b, -c$ and a after the execution of this instruction

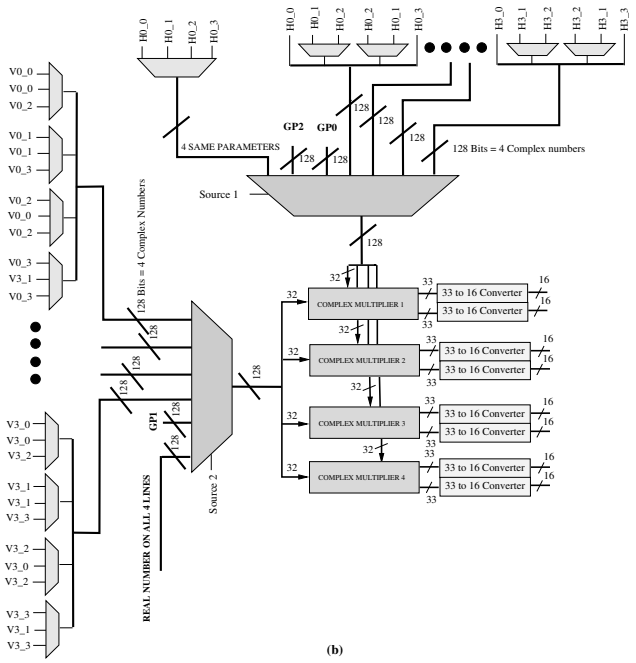
4.5.3 MULTIPLY

This category is the most demanding one in EquASIP instruction set. Different fields of the multiply instruction are detailed in Fig. 4.9(a). Eight different opcodes fall under this category to use complex multipliers for multiplication of 4×4 and 2×2 matrices (MULT4X4 and MULT2X2), multiplication of 4 complex numbers (MULT_CMPLX), 3 different MAC instructions (MAC1, MAC2 and MAC3)and two instructions to compute the output symbols $\tilde{x}$(OUT1 and OUT2). The 3×3 matrix multiplication is achieved by 4×4 matrix multiplication by providing zero at the input lines of fourth CCASM.

Different possible sources to complex multipliers are shown in the Fig. 4.9(b). Depending upon the fields “Source1” and “Source2” of the instruction, 4 operands are selected as source1 and 4 as source2 for 4 complex multipliers. To obtain different 16-bit fixed-point representations from 33-bit output of complex multipliers, 33 to 16-bit converters are designed. These converters (Fig. 4.9(c)) select 16 consecutive bits from 33-bit multiplication result depending upon the “16-Bit Control” field of the instruction. A combinational logic has also been provided to detect overflow/underflow with each choice of bit selection and consequently saturate the output value to maximum/minimum bounds. The “Destination” field of instruction selects the destination for the result.

Opcode	Source 1	Source 2	16Bit Select	Destination
4–Bit	4–Bit	4–Bit	4–Bit	4–Bit

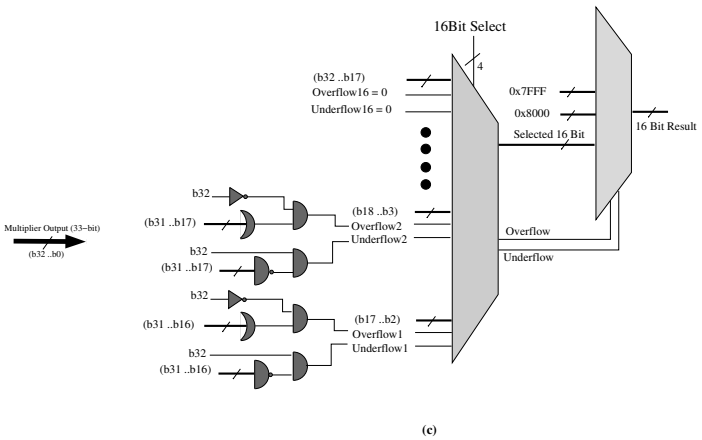
(a)



(b)

16Bit Select

33–bit Complex Multiplication Result	Binary Mapping	Syntax
32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 **17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2** 1 0	0 0 0 0	1DOT15
32 31 30 29 28 27 26 25 24 23 22 21 20 19 **18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3** 2 1 0	0 0 0 1	2DOT14
32 31 30 29 28 27 26 25 24 23 22 21 20 **19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4** 3 2 1 0	0 0 1 0	3DOT13
32 31 30 29 28 27 26 25 24 23 22 21 **20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5** 4 3 2 1 0	0 0 1 1	4DOT12
● ● ●	● ● ● ●	● ● ● ●
32 31 30 29 28 27 26 25 24 23 22 21 20 19 **18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0**	1 1 1 1	16DOT0



(c)

Figure 4.9 — Complex multiplication datapath: (a) 20-bit Multiply Instruction, (b) Possible inputs to complex multipliers, (c) 33 to 16-bit converter

4.5.4 DIVIDE

Two divide instructions have been defined. The first one is the division of a real number while the second one is used to invert a complex number. The first operation during execution of complex number division starts in the third stage of the pipeline to use the real multipliers. LUTs have been used to store the inversion values. The overall operation is shown as dotted area of Fig. 4.7.

4.6 Sample Program

In this section sample program to implement different parts of MMSE-IC2 equalization algorithm are described. The assembly code line starting with ";" designates a comment line.

4.6.1 Computation of E Matrix

The commented assembly program to compute E matrix of (2.44) is given in Listing 4.1.

Listing 4.1 — EquASIP: assembly code for E matrix computation

```
1  ;loading H matrix from channel data memory into 4
       elements of H0 MRB
2      LOAD H0_0 0x001
3      LOAD H0_1 0x002
4      LOAD H0_2 0x003
5      LOAD H0_3 0x004
6  ;loading variance of decoded symbols (x_hat) from
       input lines
7      MOVE SIG_X_TILD SIG_X_TILD_REG
8  ;Hermitian of H matrix
9      CONJUGATE OF H0
10 ;2x2 matrix multiplication to multiply H with it
       hermitian
11     MULT2X2 3DOT13 H0 V0_SU GP0_01
12     MULT2X2 3DOT13 H0 V0_SL GP0_23
13 ;loading variance of constellation used
14     LOAD SIG1 0x400
15 ;loading noise variance saved at address 0 of channel
       data memory
16     LOAD SIG_W 0x000
17 ;taking difference of variance of constellation used
       and variance of decoded symbols and saving the
       difference in SIGMA1 register
18     MOVE SIG_DIFF SIGMA1
19 ; results of multiplication of H with its hermitian
       are ready in GP0 which are copied in H0
20     MOVE GP0 H0
21 ;final E matrix computation which is
22 ;1. multiplication of each element of H0-MRB with
       SIGMA1 reg
```

```
23  ;2.addition of variance of noise at the diagonal
        positions
24  ;3.of multiplication results of step 1.
25     MAC3 1DOT15 H0 SIGMA GP0
26     NOP
27     NOP
28  ;E matrix ready in GP0
29     NOP
30  ;copying E matrix in H0
31     MOVE GP0 H0
```

4.6.2 2×2 Matrix Inversion

The commented assembly program for 2×2 E matrix inversion is given in Listing 4.2.

Listing 4.2 — EquASIP: assembly code for 2×2 E matrix inversion

```
 1  ;arranging contents of H0 (E matrix) in reverse order
        in V0
 2     MOVE_REVERSE H0
 3  ;computing determinant of E matrix
 4     MULT2X2 11DOT5 H0 V0 GP0_01
 5  ;arranging contents of H0 (E matrix) in an order
        required for matrix inversion in V0 i.e swapping
        the places of elements in first diagonal of (E
        matrix) and inverting the signs of the elements in
        the second diagonal of E matrix
 6     MOVE_MOD_COL H0
 7     NOP
 8     NOP
 9  ;taking reciprocal of determinant of E saved in GP0_0
10     DIV GP0_0
11     NOP
12     NOP
13  ;saving reciprocal of determinant of E in H0_0
14     MOVE GP0 H0
15  ;analytic method for matrix inversion
16     MULT_CMPLX 8DOT8 H0_0 V0 GP0
17     NOP
18     NOP
19  ;saving inverse of E matrix in H-MRB(0)
20     MOVE GP0 H0
```

To achieve 4×4 and 3×3 matrix inversions, above described 2×2 matrix inversion routine with 2×2 matrix addition, subtraction and multiplication instructions can be used as required in (4.5).

4.6.3 Computation of $\mathbf{p}_j$, β_j, λ_j

The parameter $\mathbf{p}_j$ of (2.45) and its hermitian can be computed using the assembly code presented in Listing 4.3.

Listing 4.3 — EquASIP: assembly code for $\mathbf{p}_j$ computation

```
1  ;loading H matrix column wise in H–MRB(1)
2     LOAD H1_0 0x001
3     LOAD H1_1 0x003
4     LOAD H1_2 0x002
5     LOAD H1_3 0x004
6     NOP
7     MOVE H1 V1
8  ;computation of pk where H0 having inverse of E and V1
       having two columns of H matrix
9     MULT2X2 6DOT10 H0 V1_SU GP0_01
10    MULT2X2 6DOT10 H0 V1_SL GP0_23
11    NOP
12    NOP
13    NOP
14    MOVE GP0 H0
15 ;computation of pk hermitian saved in MRB–V(0)
16    CONJUGATE OF H0
```

Computation of β_j is carried out as $\mathbf{p}_j^H \mathbf{h}_j$ which is shown in Listing 4.4:

Listing 4.4 — EquASIP: assembly code for β_j computation

```
1     MULT2X2 6DOT10 H1 V0 GP0_01
2     MULT2X2 6DOT10 H1 V0 GP0_23
3     NOP
4     NOP
5     NOP
6  ;saving beta in GP1 and H0
7     MOVE GP0 GP1
8     MOVE GP0 H0
```

The parameter λ_j of the expression (2.46) is computed in three steps (Lisitng 4.5): denominator is computed first, then divide instruction is used to find its inverse and finally it is multiplied with constellation variance.

Listing 4.5 — EquASIP: assembly code for λ_j computation

```
1     LOAD SIG1 0x0000
2     LOAD SIG2 0x0100
3  ;computation of denominator of lambda
4     MAC2 7DOT9 H0 SIGMA GP0
5     NOP
6     NOP
7     NOP
8  ;computation of inverse of denominator of lambda
```

```
 9      DIV  GP0_0
10      DIV  GP0_1
11      DIV  GP0_2
12      DIV  GP0_3
13      NOP
14      NOP
15      NOP
16      NOP
17      NOP
18      MOVE GP0 H0
19  ;computation of LAMBDA
20      LOAD SIG2 0
21      LOAD SIG1 0x200
22      MAC2 7DOT9 H0 SIGMA GP0
23      NOP
24      NOP
25      NOP
26      MOVE GP0 H0
```

4.6.4 Computation of $\mathbf{p}_j\lambda_j$ and g_j

These parameters are required in the computation of symbol estimation equation presented in expression 2.40. The assembly code for this computation is shown in Lisitng 4.6.

Listing 4.6 — EquASIP: assembly code for $\mathbf{p}_j\lambda_j$ and g_j computation

```
1  ;lambda1 times p1 hermitian & p2 hermitian
2      MULT_CMPLX 10DOT6 H0_0 V0_SU GP0
3  ;lambda2 times p1 hermitian & p2 hermitian
4      MULT_CMPLX 10DOT6 H0_1 V0_SL GP0
5  ;g(j) computation
6      MULT_CMPLX 10DOT6 H0 GP1 GP0
7  ;saving p1.lambd1
8      MOVE GP0 H01_01
9  ;saving p2.lambd2
10     MOVE GP0 H01_23
11     MOVE H0 V0
```

4.6.5 Symbol Estimation

Listing 4.7 is the set of instructions to find the estimated symbols:

Listing 4.7 — EquASIP: assembly code for symbol estimation

```
1  ;computation of y-H*x_hat
2      OUT1_2X2 7DOT9 H0 GP1_SU GP2_0
3      OUT1_2X2 7DOT9 H0 GP1_SL GP2_1
4  ;computation of g*x_hat
5      MULT_CMPLX 3DOT13 GP0 GP1 GP3
```

```
 6 ;loading y vector from memory
 7     LD GP2
 8     NOP
 9     NOP
10 ; computation of expression of 2.40
11     OUT2_2X2 5DOT11 GP2_SU V0 GP3
12     OUT2_2X2 5DOT11 GP2_SL V0 GP3
```

4.7 EquASIP Results and Performance

In this thesis work we used the Processor Designer tool suite from CoWare for ASIP . This tool allows to describe a processor in the LISA ADL to automatically generate the models of the processor (VHDL, Verilog or SystemC) for logic synthesis and system integration. On the other hand it provides software development tools (simulator, compiler, assembler, debugger and linker). By performing hardware synthesis and executing the application programs, performance of this ASIP is ascertained for different configurations and presented below.

Table 4.1 — EquASIP synthesis results

ASIC Synthesis Results (Synopsis Design Compiler)	
Technology	ST $90nm$
Conditions	Worst Case ($0.9V$; 105^oC)
Area	$0.37mm^2$ (84 K Gate)
Frequency	546 MHz
FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	3,174 out of 207,360 (1%)
Slice LUTs	11,299 out of 207,360 (5%)
DSP48Es	14 out of 192(7%)
Frequency	130 MHz

4.7.1 Synthesis Results

From the generated RTL description of EquASIP, logic synthesis has been conducted both on ASIC and FPGA. For ASIC target, the processor has been synthesized with Design Compiler tool from Synopsys. For FPGA target, Xilinx ISE tool has been used. In Table 4.1, the results of synthesis are summarized.

4.7.2 Execution Performance

To estimate the throughput of the EquASIP for different system configurations, the number of cycles required to compute the expressions involved in MMSE-IC1 are summarized in Table 4.2. Using this information, the user can estimate the throughput of the system under different channel's time diversity conditions and used STC. In case of quasi static conditions, after equalization coefficient computation, the throughput in terms of symbols per clock cycle is described in the last row of Table

Table 4.2 — EquASIP computation time for MMSE-IC$_1$ equations

Expression	**MIMO 2×2 (Cycles)**	**MIMO 3×3 (Cycles)**	**MIMO 4×4 (Cycles)**
E (Ref. eq. 2.44)	18	33	50
E^{-1}	14	68	68
$\mathbf{p}_j$ (Ref. eq. 2.45)	12	26	39
β_j (Ref. eq. 2.41)	7	19	27
λ_j (Ref. eq. 2.46)	23	22	23
$\lambda_j \mathbf{p}_j^H, g_j$ (Ref. eq. 2.40, 2.41)	7	12	14
Total	**81**	**180**	**221**
Symbol $\tilde{x}$ Throughput (Ref. eq. 2.40)	4 symbols/8 cycles	3 symbol/11 cycles	4 symbol/13 cycles
ASIC M Symbols/sec (@ 546 MHz)	273	149	168
FPGA M Symbols/sec (@ 130 MHz)	65	35.45	40

4.2. For a 3×3 MIMO SM configuration the symbol throughput is less than a 4×4 MIMO SM. This is due to the fact that for a 3×3 MIMO SM system 25% of the resources are not used. This illustrates a typical tradeoff between flexibility, resource utilizations and system performance. The throughput for 2×2 Golden code is same as 4×4 SM.

4.7.3 Comparison with State of the Art

In Table 4.3, different architectural parameters of state of the art implementations are summarized and compared with EquASIP implementation results. All of the referenced implementations present dedicated architecture for a specific system configuration except [77] where the proposed architecture supports 2×2 and 4×4 matrix inversion. Table 4.3 is organized in such a way that first of all comparison is made with [78], [80] and [88] which provide a complete solution to generate estimated symbol vectors. Then comparison with [75] (providing solution to compute only the coefficient matrix of (2.3)) is tabulated. Finally, the EquASIP is compared with [76], [74] and [77] which provide architectures only for matrix inversion. Furthermore, in order to make a fair comparison, the EquASIP was synthesized with the same target technology as used in the implementation with which it is being compared.

The work presented in [78] is aimed at achieving fast fading 4×4 MIMO SM using MMSE-IC. This implementation uses $\sigma_{\tilde{x}}^2 = 0$ in first iteration and $\sigma_{\tilde{x}}^2 = \sigma_x^2$ in later iterations to simplify the architecture. However, while using in iterative context this assumption of perfect $\sigma_{\tilde{x}}^2$ information induces a performance loss. Due to a fully pipelined architecture it outputs a vector containing four estimated symbols at every 38 clock cycle. Hence, the throughput is 1.31 Mega vectors at presented frequency. With EquASIP, working on same configuration, the cycles required for one symbol vector estimation are 234. This results in a throughput of 0.5 Mega vectors per second at considered frequency. Hence, the flexibility of EquASIP to support 5 different STC comes at the cost of 2.4 times less throughput, 53% more slice registers and 16 more dedicated multipliers compared to [78].

Table 4.3 — EquASIP performance comparison

Operation	System Configuration	Ref.	Algorithm	Target Device	Operate. Frequency (MHz)	Hardware Resources				Clock Cycle	Throughput (Mega Operations per sec)
						FPGA			ASIC Area (K Gates)		
						Slice/Logic Element		Dedicated Multipliers			
						Registers	LUT				
MIMO Symbol Vector Estimation	4×4 SM Fast Fading	[78]	QR CORDIC	Startix	50	8670		12	-	38	1.31
		EquASIP	Analytical		120	13272		28	-	234	0.5
	4×4 SM Block Fading	[80]	QR CORDIC	Virtex-II	140	14166		103	-	388	17.31
		EquASIP	Analytical		83	8477		14	-	845	4.71
	2×2 PC Quasi Static	[79]	Blockwise Analytical	Virtex-V	60	817	2715	60	-	1	120
		EquASIP			130	3174	11299	14	-	3.25	40
$\mathbf{p}_j$ Eq.2.3	2×2 MIMO SM	[75]	QR CORDIC	Virtex-II	-	11910		20	-	685	-
			QR SGR		-	6305		59	-	415	-
		EquASIP	Analytical		83	8477		14	-	42	1.97
	4×4 MIMO SM	[75]	QR CORDIC		-	16805		44	-	3000	-
			QR SGR		-	-		-	-	-	-
		EquASIP	Analytical		83	8477		14	-	157	0.53
Matrix Inversion	4×4 Matrix	[76]	QR SGR	Virtex-II	100	2224	2212	-	-	175	0.57
		EquASIP	Analytical		83	3177	15997	14	-	68	1.2
		[74]	QRD-RLS	Virtex-IV	115	9117		22	-	933	0.15
		[77]	Blockwise Analytical	Virtex-IV	100	1716	2094	8	-	120	0.83
				90 nm	500	-	-	-	43	92	5.43
		EquASIP		Virtex-IV	117	3232	16091	14	-	68	1.7
				90 nm	546	-	-	-	85	68	8.02

When comparing EquASIP's throughput with , In [80], the architecture implements 4×4 MIMO SM detector for 802.11n standard. In this application the design is made for a worst case scenario where for 48 vectors channel remains constant. To decode a frame of 48 vectors, the work in [80] takes 388 clock cycles. Which results in 17.3 M vectors per second at a frequency of 140 MHz. When comparing with our work, this EquASIP consumes 221 clock cycles to compute equalization coefficient for a frame and 13 clock cycles for each vector estimation. Hence the total consumed clock cycles for 48 vectors estimation are 221 + 13×48 = 845 which results in a throughput of 4.7 M vectors per second at a frequency of 83 MHz. Hence, throughput of the dedicated architecture of [80] is almost 3.6 times more at a cost of almost twice the FPGA slice used and 7.5 times more multipliers. Again this implementation is not flexible for variable antenna size, time selectivity of the channel and iterative nature of equalization.

The realization of 2×2 MMSE-IC equalizer in [79] includes pre-coding (PC). The equivalent channel matrix becomes a 4×4 matrix shown below:

$$\mathcal{H} = \begin{bmatrix} h_{11} & h_{12} & 0 & 0 \\ h_{21} & h_{22} & 0 & 0 \\ 0 & 0 & h_{11} & h_{12} \\ 0 & 0 & h_{21} & h_{22} \end{bmatrix}$$

The inversion of this matrix needs execution of two 2×2 matrix inversions. Other than this, to map this PC on the EquASIP, one 4×4 matrix multiplication is required to incorporate the PC matrix. The rest of the computations are same as required in 4×4 MIMO SM. Hence, to compute the equalization coefficients on EquASIP, 197 clock cycles will be consumed. For a target quasi-static environment, the EquASIP takes 197 cycles at 130 MHz as compared to the dedicated architecture taking 20 cycles at 61 MHz [79]. This part is not crucial because it is computed once for a frame. The throughput of EquASIP is 40 Mega symbols per second and hence 3 times less than the dedicated architecture. The 3 times faster output of dedicated architecture comes at 5 times multipliers used and this architecture used 4 times less slice registers and LUTs.

The EquASIP is better both in area and performance when compared with [75]. While comparing with [76], [74] and [77], EquASIP outperforms these architectures in throughput. EquASIP occupies more area as compared to these dedicated implementations for matrix inversion as, besides its flexibility, EquASIP supports all functions required in MMSE-IC equalization algorithm.

In the above analysis, an attempt is made to compare dedicated and flexible architectures for MMSE-based equalization. In the presence of multiple system configurations and different variants of algorithms in the equalizer, EquASIP provides a promising flexible solution compared to dedicated implementations.

4.8 Conclusion

In this chapter, the first flexible ASIP implementing an MMSE-IC linear equalizer for turbo equalization application has been presented. Analysis and simulation of mathematical equations involved in MMSE-IC LE allowed to identify potential complex-numbered operations which lead to device the instruction set for the proposed EquASIP. The specific instructions for complex number arithmetic enable to efficiently perform computations on variable sized complex numbered matrices which in turn provide required flexibility in MMSE-IC and promote its reuse for other MMSE-based applications.

Flexibility of the presented EquASIP architecture allows its reuse for each of Alamouti code, Golden code, 2×2, 3×3 or 4×4 spatially multiplexed turbo MIMO application with BPSK, QPSK,

16-QAM, and 64-QAM. When targeting 90 nm technology, the proposed architecture enables a maximum throughput of 273 MSymbol/sec for 2×2, 148 MSymbol/sec for 3×3 and 168 MSymbol/sec for 4×4 MIMO systems. The presented original contribution demonstrates promising results using the ASIP approach to implement flexible, yet efficient, MMSE-based iterative MIMO equalizer.

CHAPTER

5 DemASIP: ASIP-based Universal Demapper

THIS chapter presents the ASIP architecture dedicated for demapping function, namely DemASIP. The presented ASIP exploits the first level of parallelism available in demapping application, introduced in chapter 2 as Metric Level Parallelism. Proposed DemASIP architecture can be used for LLR generation for multiple modulation schemes adopted at the transmitter side with or without SSD . The DemASIP can work both in an iterative and a non-iterative context and provides support for BPSK to 256-QAM constellation for any mapping style used. Furthermore, DemASIP is designed to be modular in order to facilitate its integration in a scalable multiprocessor platform (exploiting the second parallelism level).

This chapter is organized in the following order. First of all, a brief state of the art section is provided to summarize the available hardware implementations related to this domain. Flexibility parameters are then analyzed to make a choice about the hardware resource allocation and sharing. The third section presents the hardware architecture for basic operators. Based on these operators, the complete DemASIP architecture together with the proposed instruction set and sample programs are presented in the subsequent three sections. Finally, the last section provides synthesis results, execution performance and comparison with state of the art implementations.

5.1 State of the Art

In the past, due to the use of low modulation order with Gray mapped constellation in wireless communication application, the part of demapping function contributes very little in a radio platform as compared to other modules such as equalizer and decoder. Most of the implementations such as the demapping function for QPSK in [79] and for QPSK and 16-QAM in [78] are based on the simplified expressions presented in [89]. The simplified expressions are only valid for Gray mapped constellation for QPSK, 16-QAM and 64-QAM. The architecture presented in [90] is also based on same approximate expressions which is only valid for the specific Gray mapped constellations of WiMax standard. A recent one, targeting DVB-T2 standard has been presented in [91]. This demapper takes into account the constellation rotation parameter associated with mapping function. The presented solution incorporates the increased demapping complexity caused by the constellation rotation which breaks the independence between the I and Q components of the QAM. Consequently, the ML QAM detector cannot apply two independent Pulse Amplitude Modulation (PAM) detectors anymore. Instead, both I and Q signal components are needed for the computation of the demapper metrics. In this case two dimensional distances as presented in (2.35) will be required. In case of small rotation angles such as required for 64-QAM and 256-QAM in DVB-T2 Standard, sub-partitioning of the constellation as presented in [91] can be applied. This reduces the search of closest constellation point complexity order from 2^m to $(2^{\frac{m-2}{2}}+1)^2$.

5.2 Flexibility Parameters and Architectural Choices

The flexibility parameters influencing the demapper architecture, implementing ML solution, comes from the expressions (2.34) to (2.37) presented in Chapter 2. The parameters to these expressions come from target constellation parameters presented in Table 1.3 and 1.4 of Chapter1. Depending upon these parameters, architectural choices can be made for efficient hardware resource allocation and sharing.

5.2.1 Flexibility Parameters

Following are the considered sources which play role in extracting the flexibility parameters towards a flexible demapper:

- Constellation definition
- Constellation sub partitioning
- Iterative demodulation

Constellation Definition: The number of bits per symbol (m) defines the constellation $\mathcal{X}$ having $M = 2^m$ symbols where each symbol represent a unique combination (binary mapping μ) of m bits. Hence m is the first flexibility parameter which comes from constellation definition. For our target ASIP m ranges from 1 to 8 for BPSK to 256-QAM respectively with any binary mapping.

Another flexibility parameter related to constellation definition is the mapping style which can be either Gray or non-Gray. In case of Gray mapped constellation without rotation and m being an even number, one can exploit the simplifications presented in 2.36 and 2.37 to compute one dimensional distances rather than the original two dimensional distances. Thus in order to exploit this simplification, the flexible demapper should efficiently support one and two dimensional distance computations.

Constellation Sub-partitioning: In order to exploit constellation sub-partitioning simplification technique another flexibility parameter can be considered. This flexibility parameter imposes the need to identify the constellation points associated to a region depending upon the sign of input received symbol.

Iterative Demodulation: The last considered flexibility parameter concerns the iterative/non-iterative nature of the demapper. In an iterative context the demapper must support the use of *a priori* information sent from the channel decoder.

5.2.2 Architectural Choices

In order to accommodate the constellation definition flexibility parameters (number of bits per symbol and their binary mapping μ on I and Q components of the symbol), a constellation LUT storing constellation parameters can be used. One possibility of achieving the flexibility is to change the content of the constellation LUT when system configuration changes. The other possible way is to save parameters of all target constellations in a single LUT where each constellation has a different stating offset address in the LUT. Hence, in this case, offset address information is sufficient to change the configuration. Our first architectural choice is based on this approach and the size of the constellation LUT is dimensioned to support the biggest target constellation, i.e 256-QAM.

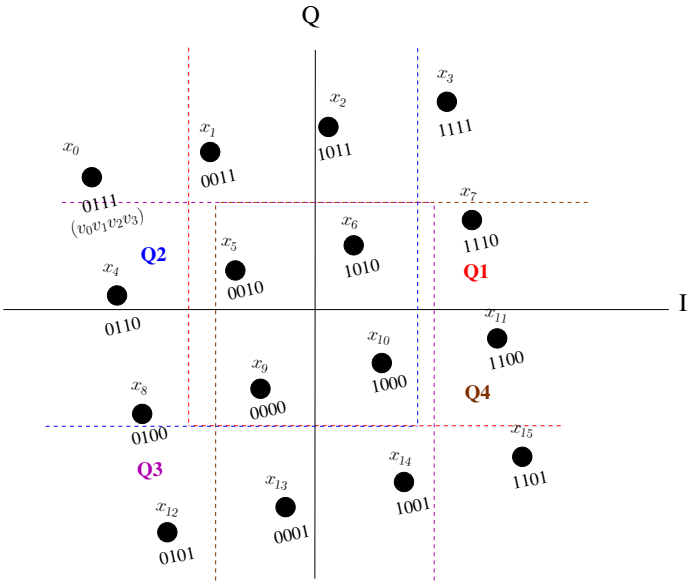
To support sub-partitioning, information of symbols related to each sub-partition can be stored at four different offset addresses in the constellation LUT. The identification to use a particular sub-partition can be provided with the help of sign bits of the received channel symbol y.

Other than constellation definition and sub-partitioning, two basic mathematical operations, Euclidean distance computation and minimum finding, are required in LLR computation expressions (section 2.2). In case of turbo demapping, add operation on *a priori* information is also required. As explained in Chapter 2, the metric level parallelism will be efficient if two distance calculators are used to compute the distance between received symbol y and two constellation points x having complementary binary mapping (Section 2.4.2). Application of this parallelism is possible when all the constellation symbols are considered. This process can be explained with the help of Fig.5.1 where a 16-QAM rotated constellation is presented with 4 sub-partitions. Consider an example of taking two Euclidean distances using x_3 and x_9, having complementary binary mapping, then with this we can update all 4 LLRs as $x_3 \in \mathcal{X}_1^i$ and $x_9 \in \mathcal{X}_0^i$ where $i = 0, 1, 2, 3$. In case of sub-partitioning implementation, it can not be guaranteed that each symbol in a sub-region with a binary mapping μ has another a point in same sub-region with complementary binary mapping μ'. In this situation only one distance calculation will be mandatory and hence resources will be under utilized. For example in $Q1$ only (x_3, x_9) and (x_1, x_{11}) exist in pair with complementary mapping. For the rest of 5 symbol i.e x_2, x_5, x_6, x_7 and x_{10}, the parallelism use will be 50% efficient.

As far as minimum operations are concerned, to support up to 256-QAM 16 minimum operations are required. By applying 16 minimum operators, parallelism efficiency decreases from 100% (for 256-QAM) to 12.5% (for BPSK) for these operations.

5.3 Hardware Architecture for Basic Operators

This section provides the details of the specific hardware components proposed to implement the architectural choices for flexibility support, established in the previous section.



$$\mathcal{X} = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}\}$$
$$\mathcal{X}_0^3 = \{x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}\} \quad \mathcal{X}_1^3 = \{x_0, x_1, x_2, x_3, x_{12}, x_{13}, x_{14}, x_{15}\}$$
$$\mathcal{X}_0^2 = \{x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}\} \quad \mathcal{X}_1^2 = \{x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7\}$$
$$\mathcal{X}_0^1 = \{x_1, x_5, x_9, x_{13}, x_2, x_6, x_{10}, x_{14}\} \quad \mathcal{X}_1^1 = \{x_0, x_4, x_8, x_{12}, x_3, x_7, x_{11}, x_{15}\}$$
$$\mathcal{X}_0^0 = \{x_0, x_4, x_8, x_{12}, x_1, x_5, x_9, x_{13}\} \quad \mathcal{X}_1^0 = \{x_2, x_6, x_{10}, x_{14}, x_3, x_7, x_{11}, x_{15}\}$$
$$\mathcal{X}(Q)_0^3 = \{1, -1\} \quad \mathcal{X}(Q)_1^3 = \{3, -3\} \quad \mathcal{X}(Q)_0^2 = \{-3, -1\} \quad \mathcal{X}(Q)_1^2 = \{1, 3\}$$
$$\mathcal{X}(I)_0^1 = \{1, -1\} \quad \mathcal{X}(I)_1^1 = \{3, -3\} \quad \mathcal{X}(I)_0^0 = \{-3, -1\} \quad \mathcal{X}(I)_1^0 = \{1, 3\}$$

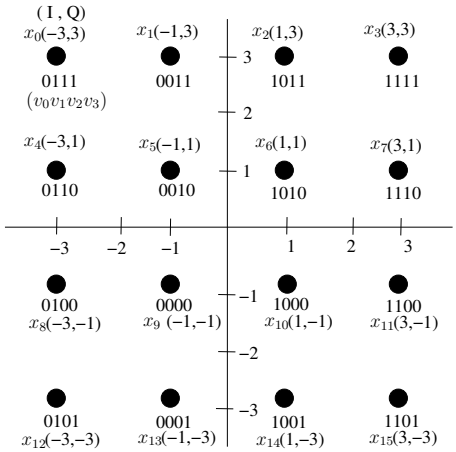
Figure 5.1 — Rotated 16-QAM Constellation with 4 sub-partitions

5.3.1 Constellation Look Up Table (LUT)

Fig.5.2 illustrates the proposed organization of the constellation LUT. In Fig.5.2(b), the constellation LUT contents represent the 16-QAM constellation of Fig.5.2(a) when Gray mapped simplifications of expressions (2.36) and (2.37) are used. Fig.5.2(c) represents the same constellation if the general demapping expression (2.35) is used. To support till 256-QAM, the μ part of the LUT will require 8 bits to store v_0 (MSB) to v_7 (LSB) and two slots will be required for I and Q parts of the symbol. Finally, to store all constellation symbols of the biggest target constellation i.e 256-QAM, 256 locations will be required. To support sub-partitioning, information of symbols related to each sub-partition can be stored at four different offset addresses. The identification to use a particular sub-partition can be provided with the help of sign bits of the received channel symbol y.

5.3.2 Euclidean Distance Calculator

For hardware efficiency over all possible scenario , we have adopted to use one distance calculator which is able to deliver one two dimensional and two one dimensional distances as shown in Fig.5.3. This unit takes channel data y, fading coefficient ρ and variance σ^2 of noise in case of single antenna transmission system. In case of MIMO this unit takes corresponding input from the equalizer as explained in Section 2.3.2. The value of x is read from the Constellation LUT described above. By visiting through the complete Constellation LUT, holding constellation information of the target modulation, all distances to generate LLRs will be available.



(a)

μ $(v_0v_1v_2v_3)$	x_I	x_Q
0000	−1	−1
0101	−3	−3
1010	1	1
1111	3	3

(b)

μ	x_I	x_Q	μ	x_I	x_Q
0000	−1	−1	1000	1	−1
0001	−1	−3	1001	1	−3
0010	−1	1	1010	1	1
0011	−1	3	1011	1	3
0100	−3	−1	1100	3	−1
0101	−3	−3	1101	3	−3
0110	−3	1	1110	3	1
0111	−3	3	1111	3	3

(c)

Figure 5.2 — 16-QAM Constellation LUT example (a) 16-QAM Constellation, (b) LUT Contents for Gray mapped simplifications, (c) LUT Contents when using Expression (2.35)

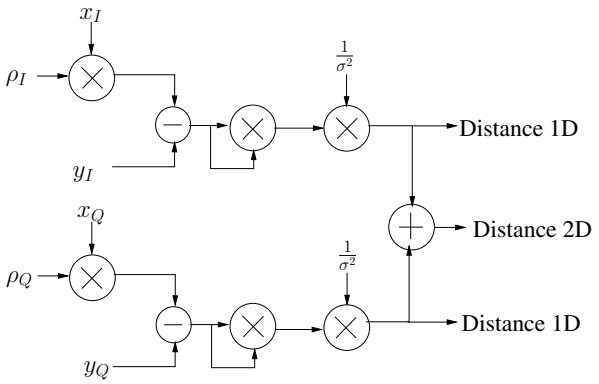


Figure 5.3 — Euclidean distance calculator

5.3.3 A priori Adder

To generate LLRs in case of turbo demodulation the *a priori* information, coming from decoders, is also used along with Euclidean distance computed from channel information as given in (2.34).

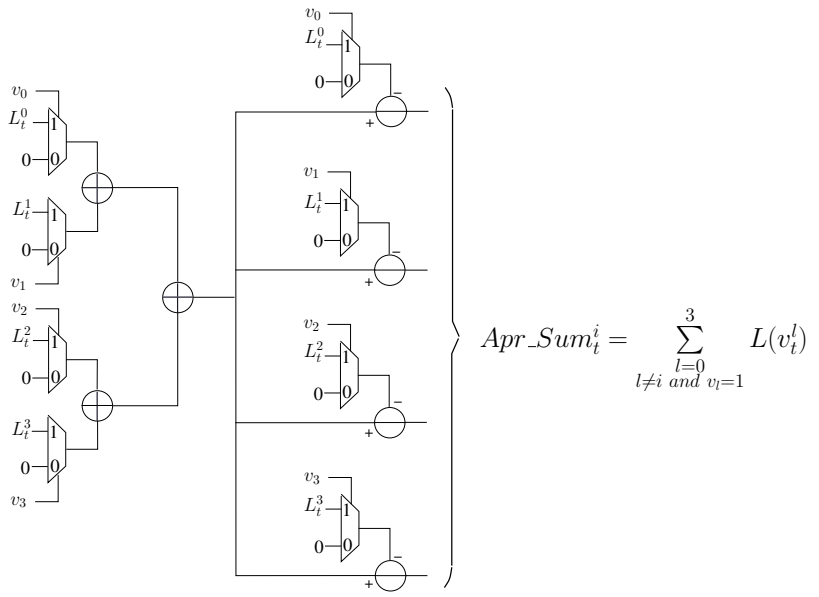


Figure 5.4 — A priori adder architecture for 16-QAM

The summation of *a priori* information for a bit i of the subject symbol x is computed by adding the corresponding LLRs of bits which are one in x and not the corresponding LLR of the bit. The implementation of this summation for 16-QAM is shown in Fig. 5.4. The input to this unit is LLRs generated by the channel decoder and the binary mapping μ from Constellation LUT.

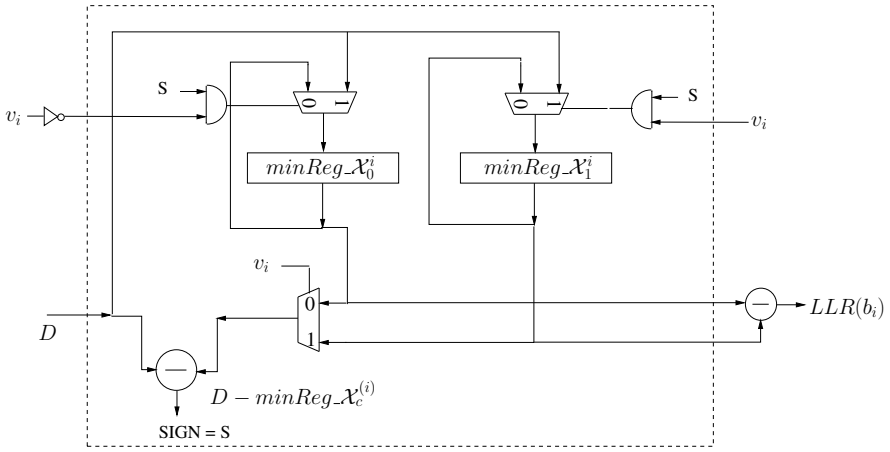


Figure 5.5 — Minimum Finder for One LLR

5.3.4 Minimum Finders

In the expressions (2.34) to (2.37), there are two minimum finding functions associated with two different symbol sets $(\mathcal{X}_0^i, \mathcal{X}_1^i)$. The idea is that when the Euclidean distance with or without *a priori* information sum is computed, the distance can be used in one of the minimum finding functions of all the m LLR computational expressions. Exploiting this property, an architecture of a minimum finder is shown in Fig. 5.5. The inputs to this unit are the Euclidean distance (D) and one of the mapping

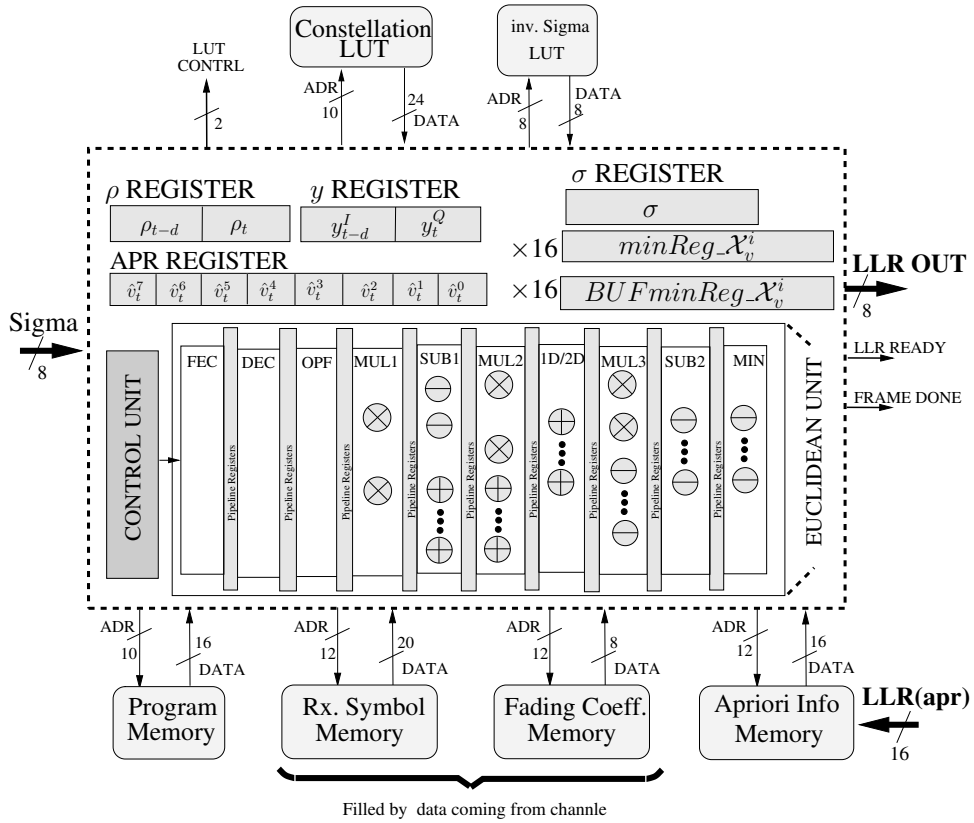


Figure 5.6 — Universal soft input soft output DemASIP architecture

bits (v_i) stored in constellation LUT. Two registers $minReg\_\mathcal{X}_0^i$ and $minReg\_\mathcal{X}_1^i$, are initialized to the maximum value at the start. Once an Euclidean distance is introduced at the input, depending upon the value of v_i, one of these register is updated if its current value is greater than the distance introduced. Total of m such units are required to support a modulation type. Hence, if a distance is computed using I and Q values of a symbol x stored in constellation LUT (Fig.5.2) at address a, the bits v_i of μ will serve as selection bits for minimum finders.

5.4 DemASIP Architecture

The proposed DemASIP architecture is constructed using one Constellation LUT, one Euclidean Distance Calculator, one a priori Adder capable of supporting 8 LLRs and eight Minimum Finders to support efficiently any constellation till 256-QAM with no restriction on mapping style, rotation angle, sub-partitioning along with turbo demodulation. The overall DemASIP diagram is shown in Fig.5.6. The input memory interface to the ASIP is comprised of "Rx. Symbol Memory" holding y, "Fading Coeff. Memory" holding ρ and "Apriori Input Memory" holding $\hat{v}_t^i$ generated by channel decoder. Two LUTs one for constellation definition and the other for $\frac{1}{\sigma_w^2}$ are also part of memory interface. The noise variance σ_w^2 is provided through a dedicated input port. The output interface is made up of LLR output, ready signals for individual LLRs availability at the output lines, frame completion and finally 2 bits to identify the received symbol location in the constellation diagram to support sub-partitioning. Besides this memory and I/O interface, DemASIP integrate registers, an Euclidean Unit (EU) and a control unit (CU). EU is spread over ten pipeline stages administered by the CU.

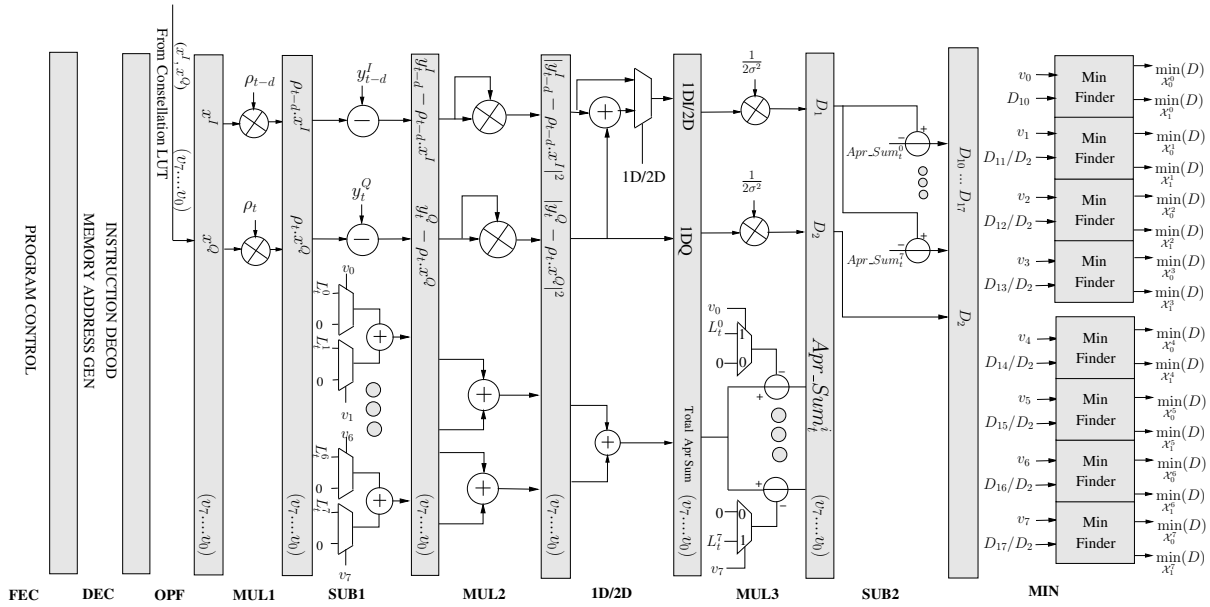


Figure 5.7 — Resource Allocation in Euclidean Unit

5.4.1 Registers

The channel data values related to a transmitted symbol i.e y and ρ are saved in their respective registers during the process of LLR generation. The *a priori* LLRs related to a symbol (up to 8 for 256-QAM) are copied in APR register. Two *a priori* LLRs are copied with one access to "Apriori Input Memory". There are 16 $minReg\_\mathcal{X}_c^i$ registers which are associated to the 8 Minimum Finders of Fig.5.5. Another set of 16 buffer registers ($BUFminReg\_\mathcal{X}_c^i$) are placed to serialize the output hence avoiding a very big output interface. In fact when parts of LLRs of a symbol are ready in the ($minReg\_\mathcal{X}_c^i$), they are copied in buffer registers and LLRs go out one by one while $minReg\_\mathcal{X}_c^i$ are used for the next symbol.

5.4.2 Euclidean Unit

The EU of DemASIP is shown in Fig.5.6. Arithmetic operators are placed in ten pipeline stages in an order to achieve all possible expressions required for LLR generation. The basic operators are distributed in different pipeline stages to reduce the critical path. Operand fetches are performed till OPF pipeline stage. The Euclidean distance calculator and *a priori* Adder are spread from SUB1 to MUL3 stage. The *a priori* Adder part only works in case when turbo demodulation context is under consideration. In 1D/2D stage the possibility of calculating one dimensional (1DI, 1DQ) and two dimensional (2D) distances is provided. The values of *a priori* information sum and Euclidean distance computed from channel data are combined in SUB2 stage. With two dimensional distance (D_1), by incorporating the *a priori* information, there are maximum eight possible distances (D_{10} to D_{17}) related to each LLR. Finally eight minimum finders are placed in MIN pipeline stage. The input to this Minimum Finders are the v_i bits of binary mapping μ and the one or two dimensional distances with or without *a priori* content.

5.4.3 Control Unit (CU)

The DemASIP control unit works as the administrator of the 10-stage pipeline EU as mentioned above and shown in (Fig. 5.6). The CU enables the data path to achieve required flexibility parameters. The functioning of the control unit will be reflected during instruction set presentation which is detailed in the next section.

5.5 DemASIP Instruction Set

The instructions of the proposed ASIP are categorized as follows.

5.5.1 Configuration Control

Configuration control instruction (SET_CONFIG) is used to configure the ASIP for different parameters such as modulation type, constellation coding, turbo/non-turbo demodulation, and SSD.

5.5.2 Input

Using different input instructions the parameters such as (y) from *Rx. Symbol Memory*, (ρ) from *Fading Coeff. Memory*, a priori information from *Apriori Info. Memory* (in turbo demodulation context), and variance of noise (σ) from *input port* are copied in respective registers. Implementation of sub-regions selection is added by analyzing the sign of I and Q components of y and then outputting two bit information to use four Look Up Tables (LUT) holding information for different sub-regions.

5.5.3 LLR Generation

The behavior of the core instruction of LLR generation (PROCESS) executes in 10 pipeline stages of the EU. It implements the LLR expressions depending upon the selected configuration. After the instruction fetch and decode, the address for Constellation LUT, storing constellation symbols x, is provided in DEC pipeline stage and data is read in OPF stage. Two one-dimensional Euclidean distances are computed using I and Q components of x, y, and ρ in MUL1, SUB1, and MUL2 pipeline stages. Two-dimensional Euclidean distance is computed by adding two one-dimensional distances in 1D/2D pipeline stage. This is required for the cases where the constellation is non-Gray or rotated and the cases where bits per symbol is odd or iterative demodulation is considered. In turbo demodulation context, depending upon the contents read from *Apriori Update LUT*, the sum of relevant a priori LLRs is computed as explained in EU description. After multiplying the distances with $\frac{1}{2\sigma^2}$ in MUL3 stage, there are three possibilities in SUB2 stage: (1) Two-dimensional distance minus a priori content, (2) Two-dimensional distance only (if *a priori* is zero), and (3) two one-dimensional distances. These distances are compared, in last pipeline stage, with the contents of one of each pair of $minReg\_\mathcal{X}_0^i$ and $minReg\_\mathcal{X}_1^i$ registers to update them for the minimum values. The minimum value update is controlled by μ value associated to x under consideration.

5.5.4 Output

Once having final values for all $minReg\_\mathcal{X}_c^i$ registers, the OUTPUT instruction transfers them to $BUFminReg\_\mathcal{X}_c^i$ registers in DEC stage and each pair is sequentially subtracted in subsequent

pipeline stages to produce LLRs at output interface. This buffering and serial transmission serves two purposes, first of all it infers smaller output interface and secondly while DemASIP sends LLRs at output, new instruction to process next symbol is launched.

5.5.5 Loop

ASIP can support one nested loop using two different types of loop instructions. First one is REPEAT instruction, which executes the set of instructions to process a symbol as many time as there are symbols in a frame. The other one is SIR (Single Instruction Repeat) which executes single instruction as many times as given in the instruction. The purpose of this instruction is to repeat LLR generation instruction for large constellations to reduce the program memory size at the cost of one cycle penalty per symbol.

5.6 Sample Program

This section illustrates the DemASIP instruction set use through two sample programs. The first sub-section demonstrates an inefficient instruction scheduling in terms of resources utilization. Avoiding this issue, through an adequate use of the proposed instruction set and hardware resources is then illustrated in the second sub-section. The assembly code line starting with ";" designates a comment line.

5.6.1 Inefficient Pipeline Usage Example

The assembly code shown in Listing 5.1 is an application program to implement (2.35) for QPSK. In this configuration the Constellation LUT has the information of four constellation symbols. After inputting variance σ^2 and setting the DemASIP configuration, the code repeats between _start and _end. This piece of code has an INPUT instruction to input y and ρ. Then four PROCESS instructions are used to generate the LLRs. Since the pipeline has 10 stages therefore one has to wait 9 cycles after the execution of last PROCESS instruction to use OUTPUT instruction. In this way of implementation, besides the required 6 instructions per symbol, there is a penalty of 9 cycles per symbol (9 NOPs) which significantly impacts the throughput.

Listing 5.1 — DemASIP: assembly code implementing inefficient pipeline usage

```
1  ;loading sigma
2          LOAD_SIGMA
3  ;settig configuration for QPSK, no Gray simplification
       , not using a priori and no SSD
4          SET_CONFIG 2bps Non Gray no Apriori No SSD
5  ;repeating code between _start and _end for 192
       symbols
6          LOAD REPEAT 191
7          REPEAT UNTIL _start _end
8  ;Inputing with no sub partition
9  _start: INPUT SINGLE LUT
10 ;Four PROCESS instruction for QPSK with 4
       constellation symbols in constellation LUT
11         PROCESS
```

```
12          PROCESS
13          PROCESS
14          PROCESS
15  ;waiting for 9 cycles for last PROCESS instruction to
        finish
16          SIR NEXT 8
17          NOP
18  ;outputting data
19          OUTPUT
20  _end:   NOP
21          NOP
22          NOP
23  ;Frame processing finished indication
24          FRAME PROCESSED
```

5.6.2 Efficient Pipeline Usage Example

The sample program shown in Listing 5.2 also implements (2.35) for QPSK but with this code there is no need to wait for LLR to be ready. Instead, instructions to process next symbol is launched in pipelines and after three symbols the program code is repeated for the coming symbols. At the end there will be few NOPs for the LLRs of last two symbols of a frame (total 9 NOPs per frame). In this way the programmer saves the penalty of 9 cycles per symbol as no NOP is required.

Listing 5.2 — DemASIP: assembly code implementing efficient pipeline usage

```
 1          LOAD_SIGMA
 2          SET_CONFIG 2bps Non Gray no Apriori No SSD
 3          LOAD REPEAT 188
 4          REPEAT UNTIL _start _end
 5  ;first symbol
 6          INPUT SINGLE LUT
 7          PROCESS
 8          PROCESS
 9          PROCESS
10          PROCESS
11  ;second symbol
12          INPUT SINGLE LUT
13          PROCESS
14          PROCESS
15          PROCESS
16          PROCESS
17  ;third symbol
18          INPUT SINGLE LUT
19          PROCESS
20          PROCESS
21  ;LLRs ready for first symbol
22          OUTPUT
23          PROCESS
24          PROCESS
```

```
25 ;fourth and next symbol
26 _start:  INPUT SINGLE LUT
27          PROCESS
28 ;LLRs ready for second and next symbols
29          OUTPUT
30          PROCESS
31          PROCESS
32 _end  :  PROCESS
33          SIR NEXT 2
34          NOP
35 ;outputting LLRs for second last symbol
36          OUTPUT
37          SIR NEXT 5
38          NOP
39 ;outputting LLRs for last symbol
40          OUTPUT
41          NOP
42 ;Frame processing finished indication
43          FRAME PROCESSED
```

The same program can be used for the 16-QAM Gray mapped constellation having Constellation LUT contents shown in Fig.5.2(a) by changing the SET_CONFIG instruction as follows:

```
SET_CONFIG 4bps Gray no Apriori No SSD
```

By this change expressions (2.36) and (2.37) will be executed for 16-QAM constellation.

5.7 DemASIP Results and Performance

As for EquASIP, we used the Processor Designer tool suite from CoWare to implement the proposed DemASIP. This tool allows to describe a processor in the LISA ADL to automatically generate the models of the processor (VHDL, Verilog or SystemC) for logic synthesis and system integration. On the other hand it provides software development tools (simulator, compiler, assembler, debugger and linker). By performing hardware synthesis and executing the application programs, performance of this ASIP is ascertained for different configurations and presented below.

5.7.1 Synthesis Results

From the generated RTL description of DemASIP, logic synthesis has been conducted both on ASIC and FPGA. For ASIC target, the processor has been synthesized with Design Compiler tool from Synopsys. For FPGA target, Xilinx ISE tool has been used. Results of synthesis are summarized in Table 5.1.

5.7.2 Execution Performance

The throughput of the DemASIP under different system configurations is summarized in Table 5.2. The number of cycles for the demapping of one symbol are calculated as one cycle each for INPUT

Table 5.1 — DemASIP synthesis results

ASIC Synthesis Results (Synopsis Design Compiler)	
Technology	ST $90nm$
Conditions	Worst Case ($0.9V$; 105^oC)
Area	$0.1mm^2$ (23.5 K Gate)
Frequency (f)	537 MHz
FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	1,918 out of 207,360 (1%)
Slice LUTs	3,201 out of 207,360 (1%)
DSP48Es	6 out of 192(3%)
Frequency (f)	186 MHz

Table 5.2 — DemASIP execution performance results

Modulation	**Bits per Symbol** (m)	**Clock Cycles to demapp one Symbol** (c)	**Throughput** (MLLRs/sec)	
			ASIC	**FPGA**
For Gray Mapped Simplified Expression (2.36 and 2.37)Implementation				
QPSK (Gray)	2	4	269	93
16-QAM (Gray)	4	6	358	124
64-QAM (Gray)	6	10	322	112
256-QAM (Gray)	8	18	239	83
For (2.34 or 2.35) Implementation				
QPSK	2	6	179	62
8PSK	3	10	161	56
16-QAM,16APSK	4	18	120	42
32-APSK	5	34	79	27
64-QAM	6	66	49	17
64-QAM (using sub partitioning)	6	27	119	41
256-QAM	8	258	17	6
256-QAM (using sub partitioning)	8	83	52	18

and OUTPUT instructions and rest of the cycles are those when PROCESS instruction is launched (which is equal to the number of constellation entries used from Constellation LUT). Few cycles required for instructions such as configuration instruction, loop instruction in the start of the application program and NOPs at the end of the program are neglected. The throughput in Mega LLR/sec is computed as follows:

$$Throughput = \frac{f}{c} \times m \tag{5.1}$$

Table 5.3 — DemASIP results comparison

	Ref. [90]	**Ref. [91]**	**DemASIP**
Wireless Standard	Wimax	DVB-T2	WiFi, Wimax, UMB, LTE, DVB-SH,S2,T2
Modulation Support	QPSK,16-QAM 64-QAM (Gray Mapped Wimax Constellation)	QPSK,16-QAM 64-QAM, 256-QAM (DVB Constellation)	From BPSK to 256-QAM (Any Constellation) and Iterative Demapping
Frequency (MHz)	224	62	186
Throughput 64-QAM	MLLR/sec	MLLR/sec	MLLR/sec
-Gray	224	-	112
-With SSD (Sub partitioning)	-	124	41
Area			
-Slice Registers	741	791	1,918
-Slice LUTs	378	4,667	3,201
-DSP48Es	0	20	6
-BRAM	0	0	8

5.7.3 Comparison with State of the Art

Result comparison with state of the art solutions has been summarized in Table 5.3 as per available information. The flexibility of [90] is limited not only to three types of modulation but also to Gray mapped Wimax constellation without the support for iterative demodulation. Due to approximate LLR computation expressions and very limited flexibility, the architecture achieves better frequency and throughput as compared to our solution. Similarly, the demapper proposed in [91], targets the DVB-T2 parameters and does not support iterative demapping. Their dedicated architecture, computing 9 Euclidean distances per cycle using 9 parallel computation units, achieves high throughput at the cost of high hardware resource occupation. In fact, our proposed DemASIP architecture constitutes the first solution to the best of our knowledge, where it provides full flexibility to support any modulation adopted in multiple wireless standards with reduced hardware resource occupation. The salient features of DemASIP, such as low area and high flexibility, promise its use to address low and high throughput demands using single and multi-ASIP architectures.

5.8 Conclusion

In this chapter, we have presented the first flexible ASIP implementing a universal demapper for multi wireless communications standards. With analyzed LLR computational expressions presented in chapter 2, flexibility parameters and common operators are identified. These flexibility parameters are then assigned to efficient hardware components and a specialized instruction set has been designed. The architecture proposed addresses all the complexity levels associated with demapping functionality through the arrangement of certain LUTs and EU which allows its reuse both in an iterative and a non-iterative context and provides support for BPSK to 256-QAM constellation for any

mapping style used. The specialized instruction set provides liberty to generate the LLRs in different system configurations. When targeting 90 nm technology, the proposed architecture enables a maximum throughput of 358 Mega LLRs per second for 16-QAM Gray mapped constellation.The presented original contribution demonstrates promising results using ASIP approach to implement the universal demapper.

CHAPTER

6 Multi-ASIP NoC Based Turbo Receiver

THIS chapter is dedicated to the presentation of the prototyping flow of the individual proposed ASIPs and the proposed heterogeneous multi-ASIP NoC based architecture for turbo reception. In the first part of the chapter LISA to FPGA prototyping flow is detailed. Based on this flow, design, validation and prototyping of EquASIP and DemASIP (presented in the two previous chapters) are described. Besides individual component verification, proof of concept of multi-ASIP and NoC based architectural solution towards a unified turbo receiver is another important aspect of this thesis. To that end, three incremental complexity multi-ASIP prototypes have been realized and will be presented in last three sections of this chapter. The first one demonstrates parallel turbo decoding, the second demonstrates parallel turbo demodulation and decoding and the third one demonstrates parallel turbo demodulation, equalization and decoding. The proposed multi-ASIP architectures demonstrate the efficient exploitation of the second level of parallelism available in turbo equalization, demodulation and decoding applications, introduced in chapter 2 as Component Level Parallelism (sub-blocking and shuffled parallelism techniques). For all of these three multi-ASIP prototypes, hardware implementation of functional blocks of transmitter, channel and receiver are described. With each prototype the results of synthesis and FER results, for different system configurations, acquired from FPGA platform are also presented.

6.1 ASIP Design, Validation and Prototyping Flow

While selecting ASIP as the implementation approach, an ASIP design flow integrating hardware generation and corresponding software development tools (assembler, linker, debugger, etc.) is mandatory. In our work we have used Processor Designer framework from Coware Inc. which enables the designer to describe the ASIP at LISA [53] abstraction level and automates the generation of RTL model along with software development tools. ASIP design, validation and prototyping flow has been divided into 3 levels of abstraction as shown in Fig.6.1 and is detailed in the following subsections.

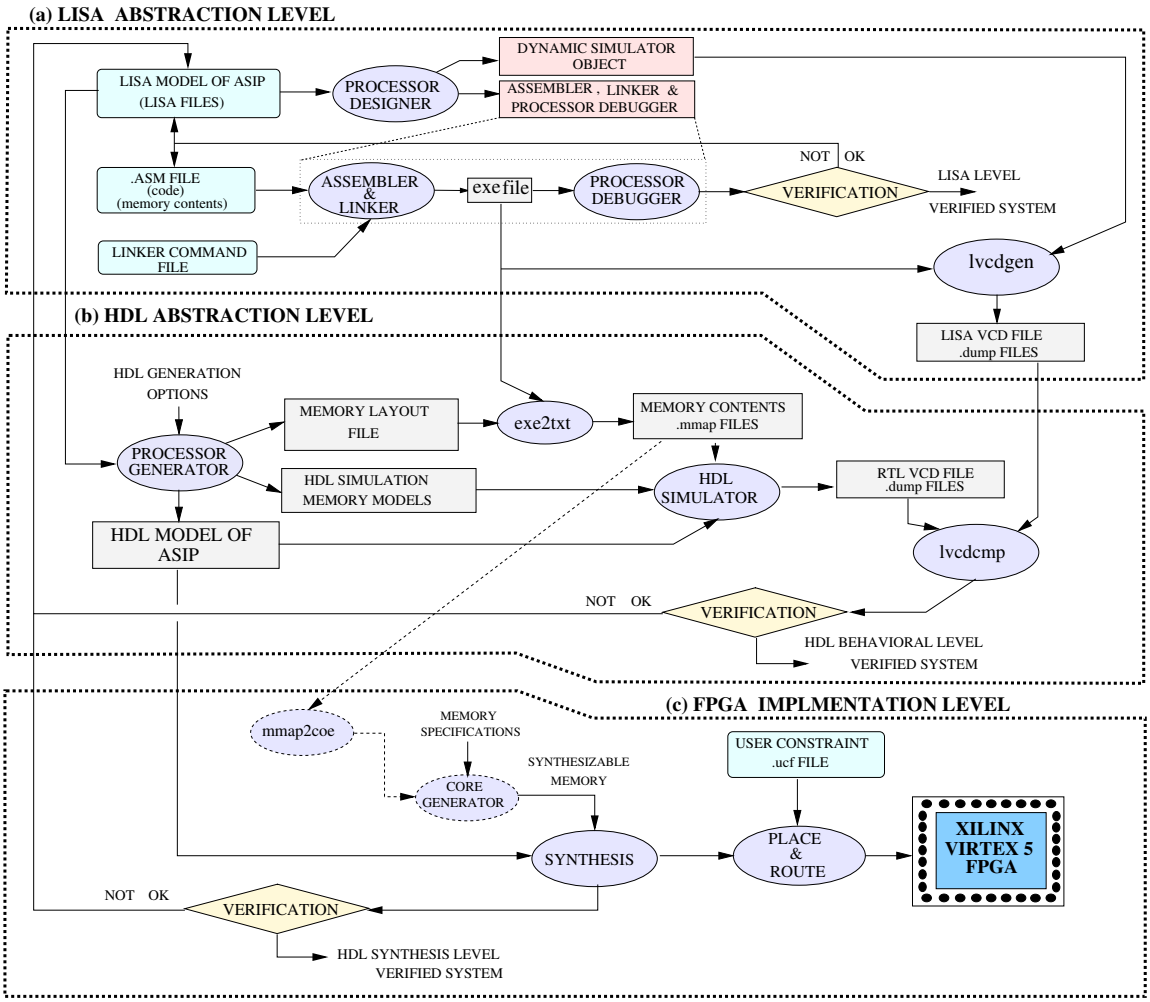


Figure 6.1 — Prototyping Flow: (a) LISA abstraction level, (b) HDL abstraction level, (c) FPGA implementation level

6.1.1 LISA Abstraction Level

The first step towards the ASIP implementation is the LISA ADL modeling of the proposed architecture and the application program writing (.asm file) to be executed on the ASIP. To simulate the input data memories the contents of these memories, taken from the software reference model of the target application, are written in different sections of the assembly file as defined in the linker command file. With ADL model of the ASIP, Processor Designer framework generates tools like assembler, linker,

processor debugger and simulator. Assembler and linker process the application program (.asm file) to generate the executable file (.out file) which is used in Processor Debugger to verify both the ASIP model and the application program. Once the ASIP is verified, a special utility "lvcdgen" can be used to generate Value Change Dump (VCD) file which store all registers content and ASIP output values during the application program execution. The generated VCD file can be used at lower abstraction levels for verification purpose. The "lvcdgen" utility uses Dynamic Simulator Object and executable file of the application to produce this reference VCD file. The complete flow is shown in Fig.6.1(a).

6.1.2 HDL Abstraction Level

Processor Designer framework provides the Processor Generator tool which is configured to generate HDL (VHDL/Verilog) model of the ASIP from LISA model, simulation models of memories and the memory layout file as shown in figure Fig.6.1(b). The quality of the generated HDL depends upon the LISA modeling and the configuration options of Processor Generator. It is highly recommended that LISA modeling should be as close as possible to HDL, e.g if in one pipeline stage we want resource sharing, that resource should be declared once. Otherwise, due to inability to detect sharing, resources will be duplicated in HDL. Other issue is the use of high level operators of LISA which may not be produced by the Processor Generator e.g modulo two operation ("variable $\%$ 2" in LISA) should be rather implemented by the LSB manipuation of the considered variable. For memory interface generation, different Memory Interface Definition Files (MIDF) are provided which define the number of ports and latencies.

Once memory layout file and executable application program file is available, "exe2bin" utility inputs them to generate the contents of memories in separate .mmap files. With these three inputs (VHDL model, memory model and .mmap files), the VHDL model can be simulated behaviorally using an HDL simulator, e.g ModelSim by Mentor Graphics.

To run HDL simulation, Processor Generator produces ready-to-use Makefile which can be executed to see either the waveforms or to generate VCD file. To verify the generated ASIP HDL model, the VCD file generated through HDL model and the one generated through LISA model (in previous subsection) can be compared using "lvcdcmp" utility.

6.1.3 FPGA Implementation Level

At this level, the only missing elements are the synthesizable memory models. Depending upon the FPGA selected, equivalent synthesizable memories are generated through FPGA vendor specific tools and at the same time .mmap memory content files have to be translated, if necessary, in required format for compatibility. With Xilinx devices,"Core Generator" tool can be used to generate the synthesizable memories and "mmaptocoe translator" converts .mmap files into required .coe format. With this complete synthesizable HDL model, synthesis can be performed as shown in Fig.6.1(c). After successful synthesis, the placement and routing is performed as per the user constraints file (.ucf file). Inside .ucf file, the user inputs the platform dependent timing and location constraints, e.g the operational frequency and input/output pins. The final step is the generation of the configuration file which can be used to configure the FPGA for the final ASIP prototype model.

6.2 EquASIP and DemASIP FPGA Prototyping

On board validation is a crucial step in order to demonstrate the feasibility, resolve any eventual system and/or environment issue, and measure the exact performance of the designed architecture. In

our case, a logic emulation board (DN9000K10PCI) integrating 6 Xilinx Virtex 5 devices was available and has been used to validate the designed ASIPs. With this board, appropriate communication controllers are available and can be added to the design in order to read/write various output/input memories from a host computer using a USB interface.

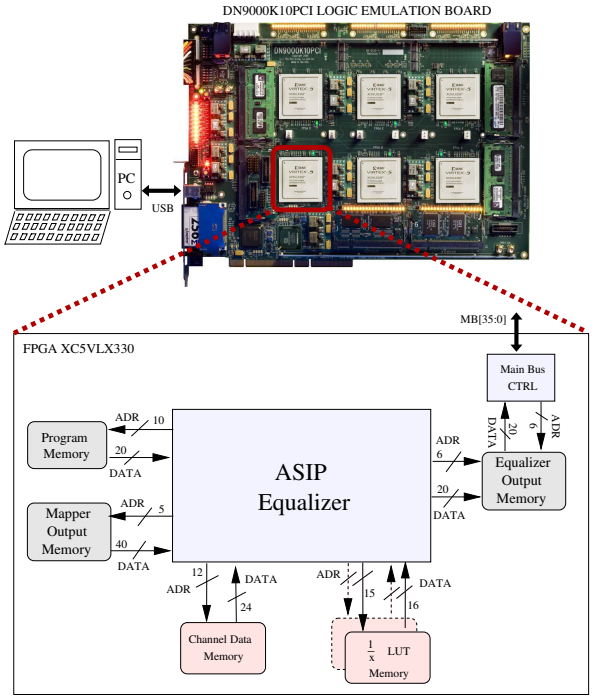


Figure 6.2 — EquASIP on-board prototype

6.2.1 EquASIP FPGA Prototype

Using the Xilinx tool suite ISE, a new project was created integrating the ASIP, corresponding memories, and a board communication controller as shown in Fig.6.2. The contents of the input memories i.e Channel Data Memory, $\frac{1}{x}$ LUTs and Mapper Output Memory were generated automatically from the fixed-point software reference model in .coe file format along with a reference result file containing the output of the equalizer. In this prototype, except Channel Data Memory and $\frac{1}{x}$ LUT which are synchronous, rest of the memories are asynchronous. Xilinx Virtex 5 device provides two type of memories, Distributed and Block Memories which can be customized for asynchronous and synchronous respectively. In order to record ASIP's results and to compare them with reference result file, a dual port Equalizer Output Memory has been created. One port of this memory is written with equalization results from EquASIP side and the other port is read by external host computer through USB interface. On this host computer, a graphical user interface with adapted parameters is used in order to setup the various parameters of the board and to download the output memory contents for comparison with reference result file.

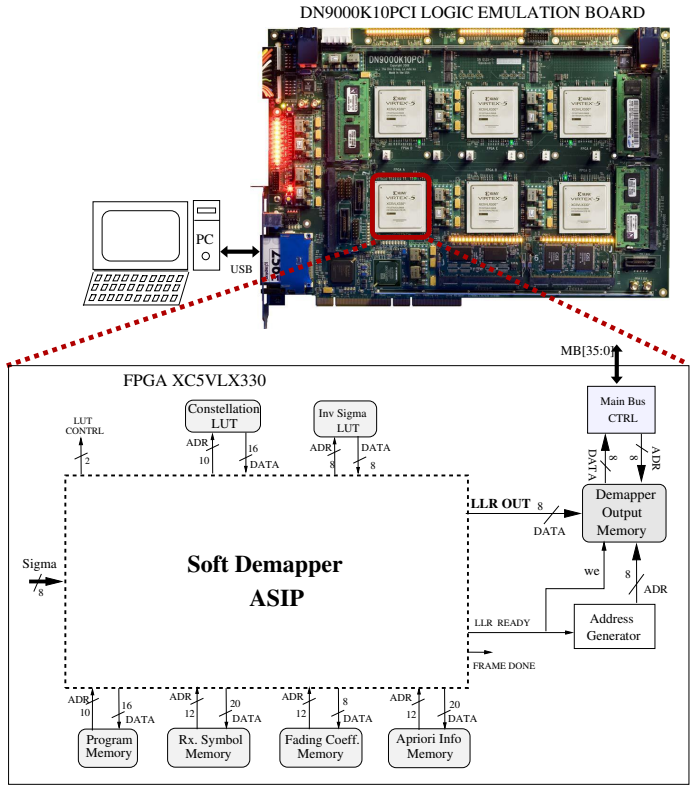


Figure 6.3 — DemASIP on-board prototype

6.2.2 DemASIP FPGA Prototype

Using the same principle as adopted in EquASIP, the contents of the inputs memories to the DemASIP were generated from quantized reference software model and the output results of the DemASIP were compared with reference software output results. The prototype diagram of DemASIP is shown in Fig.6.3.

Besides individual component verification, proof of concept of multi-ASIP and NoC architecture implementing a unified turbo receiver is another important aspect of this thesis. To that end, three incremental complexity multi-ASIP prototypes have been realized. The first one demonstrates parallel turbo decoding, the second demonstrates parallel turbo demodulation and decoding and the third one demonstrates parallel turbo demodulation, equalization and decoding. These three multi-ASIP prototypes will be presented in the three following sections.

6.3 First multi-ASIP Prototype: Parallel Turbo Decoder

In this section we present the first multi-ASIP prototype realized to demonstrate parallel turbo decoding. A first effort towards this objective has been started through the work presented in [39]. The proposed prototype at that time was missing few components (code rate control, BICM and higher order modulation functionalities) for its integration towards a unified turbo receiver. To add these functionalities, on transmitter side the puncturing, bit interleaving and mapping functions are designed and integrated. Channel is modified to include the fading effects and hence Rayleigh fading

channel is implemented. On the receiver side demapping, bit deinterleaving, depuncturing is added with decoding functions. The target system block diagram for this first multi-ASIP prototype is shown in Fig.6.4.

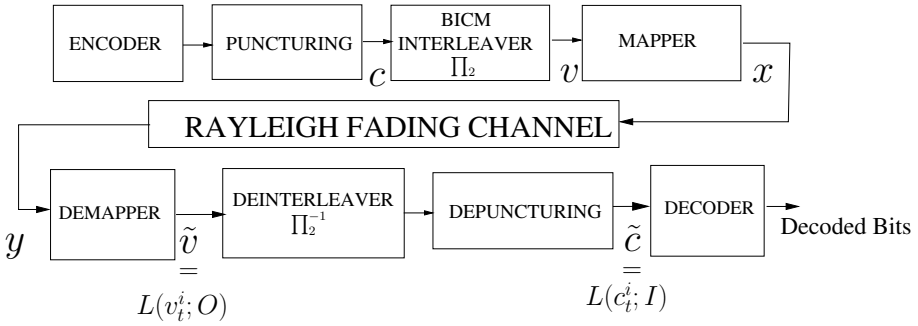


Figure 6.4 — Turbo coded transmission system diagram

6.3.1 Transmitter

The specifications of the transmitter system are taken from Wimax standard and the hardware architecture of each sub-system is given below.

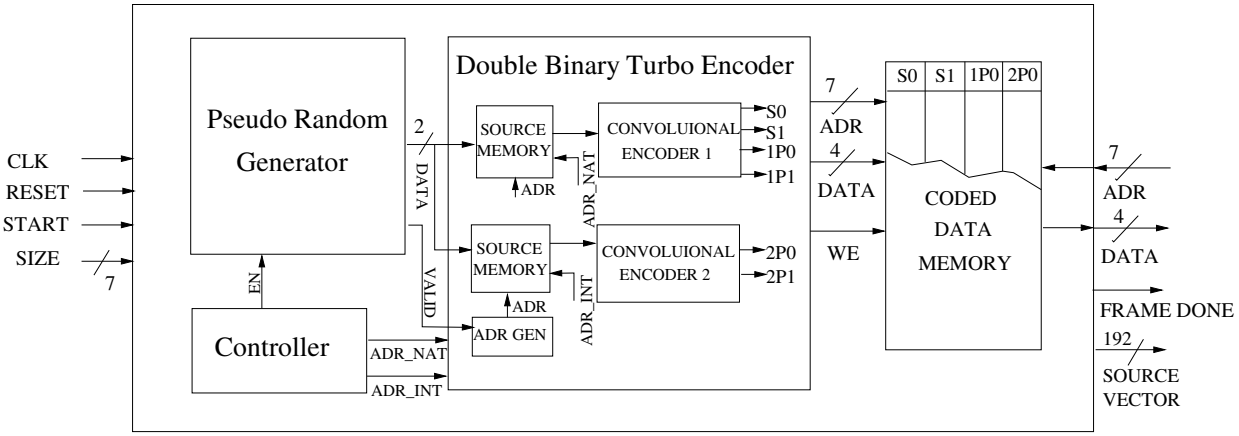


Figure 6.5 — Turbo encoder with random source generator

6.3.1.1 Encoder

The first element of the transmitter is shown in Fig.6.5. This block includes random bit generator, two encoders of double binary turbo codes and a controller. For this platform the encoder is configured for 24 byte random source which is generated by a Pseudo Random Generator unit after receiving the START pulse. Each of the 2-bit source symbol are saved in two source memories. After this source generation, the controller enables the encoder block and provides the natural and interleaved addresses (using a LUT storing the interleaving table of used turbo code) to the source memories on their read ports. Using 2-bit source symbol, both in natural and interleaved order, each encoder generates two parity signals. To support code rates greater and equal to 0.5 given in Wimax standard, two source

bits (S0,S1) and one parity from each encoder is copied in the Coded Data Memory. At the end of the encoding, the controller puts all the components in idle state and generates an active high pulse on FRAME DONE output line. This pulse can be used to start the next process which can use encoded data by reading the Coded Data Memory. To change the frame size one needs to give the frame size information on SIZE input port and updates the interleaving LUT with the corresponding values. Source bit vector is also provided at the output for its comparison with receiver's output results for error rate performance computation. The FPGA synthesis results of this encoder are shown in Table 6.1.

Table 6.1 — Synthesis results of encoder block

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	505 out of 207,360 (<1%)
Slice LUTs	163 out of 207,360 (<1%)
DSP48Es	0 out of 192(0%)
Frequency (f)	300 MHz

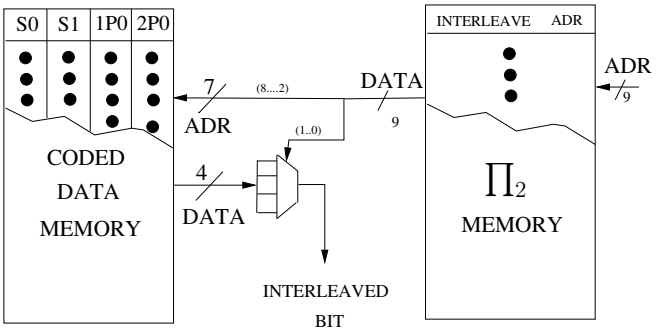


Figure 6.6 — Puncturing and BICM interleaving

6.3.1.2 Combined Rate Control and BICM Interleaver

The puncturing and BICM interleaving are achieved using a LUT, called $\prod_2$ Memory, storing the BICM interleaving table with puncturing details. The process is shown in Fig.6.6. To retrieve an interleaved bit, the corresponding interleaved address is read from $\prod_2$ Memory. This address has two parts, the 7 MSBs (2 to 8) represent the coded symbol index whereas the 2 LSBs (0 and 1) are used to select one of the bits in the coded symbol to achieve puncturing along with interleaving. One needs to change the contents of the $\prod_2$ Memory to support other configurations. The hardware cost of this element is a memory having 512 address spaces of 9-bits. In the system under study the contents of $\prod_2$ Memory are generated using (1.6 and 1.7).

6.3.1.3 Parametrized Mapper

The mapping function is achieved by designing a parametrized mapper as shown in Fig.6.7. The parameters provided to this hardware block are bits per symbol m, the number of symbols to map and information about Gray or non-Gray mapping style. To support up to 256-QAM, 4 bits are reserved

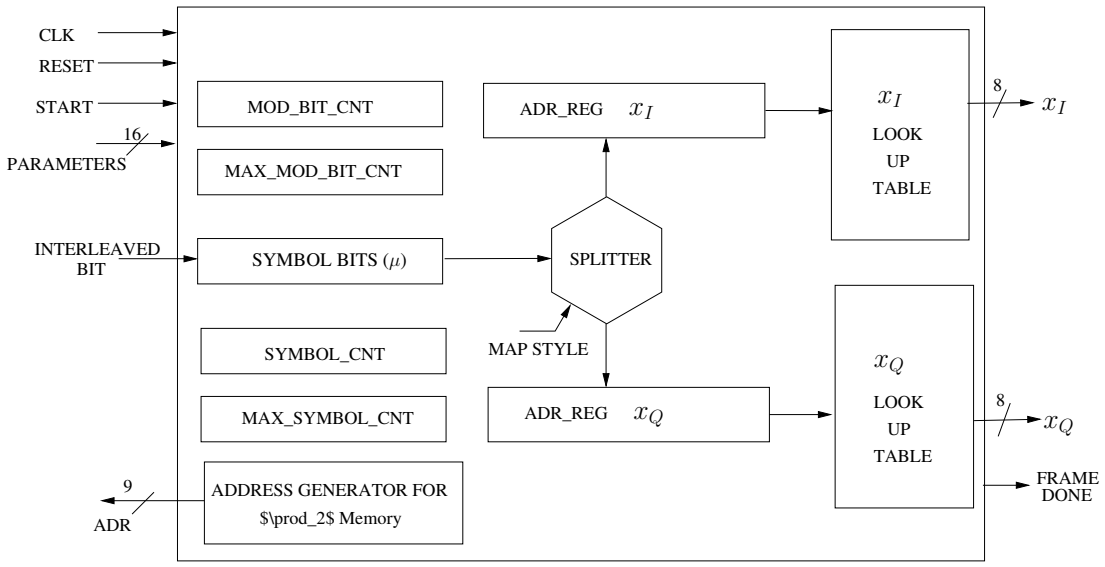


Figure 6.7 — Parametrized mapper block diagram

to represent m. 11 bits are reserved to indicate the number of symbols to map. Finally 1 bit is used to indicate mapping style.

With reset pin at zero and a valid clock edge, the mapper starts up in idle state. On receiving active high pulse on start pin, the parameters such as m is saved in MAX_MOD_BIT_CNT register and the number of symbols to map in MAX_SYMBOL_CNT register. The value to be stored in MAX_SYMBOL_CNT register can be computed using the number of source bits (s), code rate r and m as follows:

$$Max.\ Symbols = \frac{s}{r \times m} \tag{6.1}$$

After parameter saving, the mapper enters in processing phase which is comprised of bit gathering state and symbol construction state. In bit gathering state, to acquire an interleaved bit, an address is sent to $\prod_2$ Memory of Fig.6.6 which generates the interleaved address for the required bit saved in Coded Data Memory. The acquired bit goes in SYMBOL_BIT (μ) register and the MOD_BIT_CNT register is incremented. This bit gathering process continues till MOD_BIT_CNT is equal to MAX_MOD_BIT_CONT register, i.e equal to m. Using these symbol bits, in case of Gray mapped constellation, the splitter sends half of the bits on the address lines of x_I LUT and the rest half on the address lines of x_Q LUT to have a modulated symbol. In case of rotated, non-Gray or m is an odd number, all gathered bits are sent both to the x_I and x_Q LUTs. After each symbol mapping, SYMBOL_CNT register is incremented by one and mapping process continues till SYMBOL_CNT register is equal to MAX_SYMBOL_CNT register. After the mapping of all the symbols, this parametrized mapper outputs a FRAME DONE pulse and goes back to idle state. The FPGA resource used for this block are summarized in Table 6.2. In the system under study the mapper is configured for QPSK and a total number of 192 symbols.

6.3.2 Rayleigh Fading Channel

To emulate a Rayleigh fading transmission channel, a hardware model has been developed in the Electronics Department of Telecom Bretagne which is used in this prototype. This generic channel

Table 6.2 — Synthesis results of parametrized mapper block

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	127 out of 207,360 (<1%)
Slice LUTs	130 out of 207,360 (<1%)
DSP48Es	0 out of 192(0%)
Frequency (f)	336 MHz

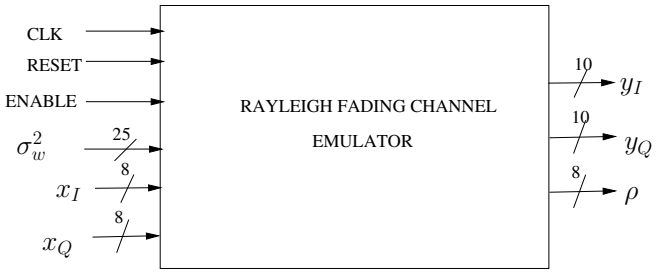


Figure 6.8 — Rayleigh Fading Channel Emulator

model can be used both in single antenna and MIMO channel emulation. For single antenna configuration the I/O interface of this channel model is shown in Fig.6.8. Besides clock and reset, channel model takes modulated symbol $x(I, Q)$ with its enable signal (showing its availability on input line) and noise variance σ_w^2. $y(I, Q)$ and fading coefficient ρ are the output of this module which are saved in Rx. Symbol Memory and Fading Coeff. Memory respectively. The synthesis results of this channel emulator are tabulated in Table 6.3.

Table 6.3 — Synthesis results of Rayleigh fading channel block

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	4532 out of 207,360 (2%)
Slice LUTs	4406 out of 207,360 (2%)
DSP48Es	30 out of 192(15%)
Frequency (f)	65 MHz

6.3.3 Receiver

The receiver part is comprised of DemASIP used for demapping function, combined architecture for BICM deinterleaving and depuncturing, and finally multiple TurbASIP interconnected by means of two unidirectional Butterfly NOC for turbo decoding.

6.3.3.1 DemASIP Integration

The DemASIP of Fig.5.6 is used in the receiver for demapping function. The ASIP is configured to implement Gray mapped simplified expression (2.36) and (2.37) with no SSD and no turbo demodulation context for the demapping of QPSK symbol. Using DemASIP, to demap a Gray coded QPSK

modulated symbol, 4 clock cycles are required to generate two LLRs as given in 5.2. Hence a total of 768 clock cycles are required for the demapping of 192 symbols. Additional 16 cycles are required for system configuration in the start of demapping and NOP instructions at the end of the demapping process.

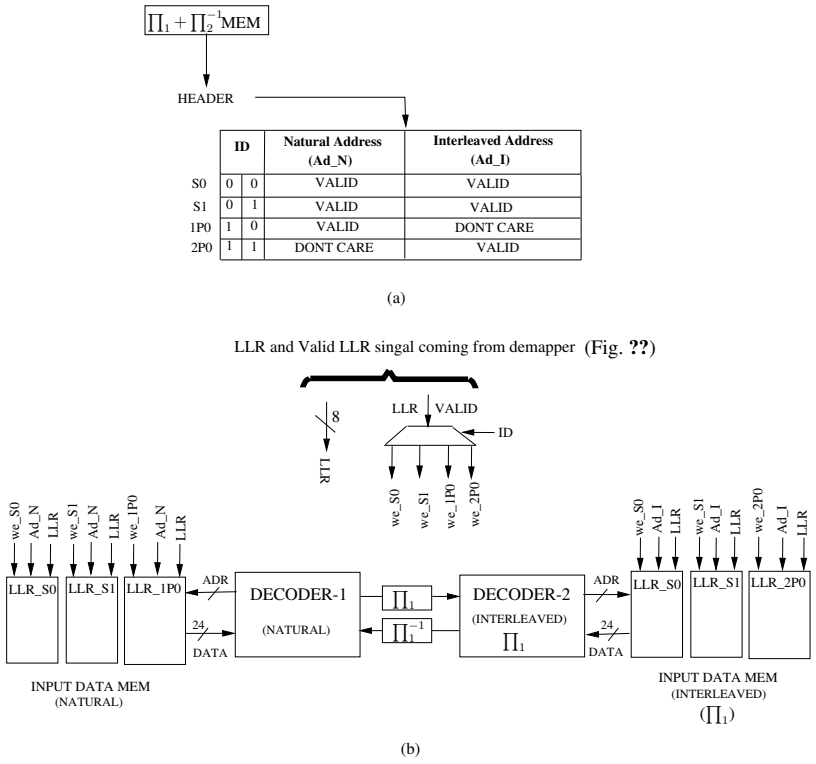


Figure 6.9 — BICM deinterleaving and depuncturing implementation (a) $\prod_2^{-1}$ and header (b) Decoder memory address decoding

6.3.3.2 BICM Deinterleaving and Depuncturing

After having LLRs at the output of DemASIP, the next step is to implement deinterleaving and depuncturing function. To implement this function, one needs to first identify the bit (which can be either source or parity) and then to identify the location of this LLR in the coded symbol block. In case the LLR is related to source bit then it will go both to the memory of the decoder working in natural domain and the memory of decoder working in interleaved domain ($\prod_1$). Hence for source bits BICM deinterleaving ($\prod_2^{-1}$) and ($\prod_1$) will be applied. In case of LLR related to parity, the LLR will be stored in the memory attached to the ASIP which is either working in natural domain or the one which is working in interleaved domain (not both).

To implement this, like other interleaving/deinterleaving functions, precomputed LUT stored in $\prod_2^{-1} + \prod_1$ Mem. is used. Once LLR is available at the output, a header coming from $\prod_2^{-1}$ Mem. is attached with this LLR as shown in Fig. 6.9(a). This header contains first two bits reserved to identify if the LLR is related to S0, S1, 1P0 or 2P0. Two addresses, adr_N and adr_I, are provided in next two fields of the header. The first one contains the memory address for the input data memory of the decoder working in natural domain whereas the other is for the memory related to the decoder working in $\prod_1$ domain. In case the LLR is related to the parity of the first encoder, only adr_N will

be valid and vice versa for the LLR related to the second encoder working in $\prod_1$ domain as shown in Fig.6.9(b).

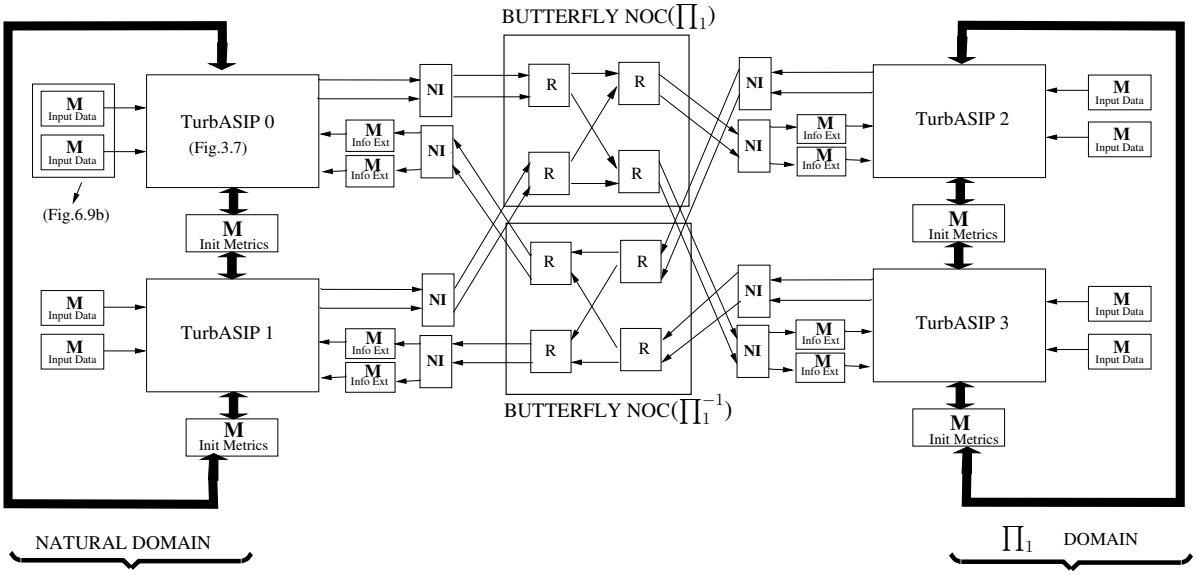


Figure 6.10 — Multi-ASIP and Butterfly NoC architecture for turbo decoding

6.3.3.3 multi-ASIP and NoC Based Turbo Decoder

Initially, the TurbASIP presented in Fig.3.7 is designed to take 4-bit LLR to support BPSK. To make this ASIP capable to support 256-QAM, the input interface of the ASIP is modified to handle the 8-bit LLR generated by the demapper. Consequently the datapath is proportionally increased to support increased input data quantization. A turbo decoding platform composed of 4 TurbASIP and 2 unidirectional Butterfly NoC, shown in Fig.6.10, is automatically generated using the Auto Generation Tool presented in the thesis work of [43]. In this platform, each ASIP takes 7 cycles to produce extrinsic information (or hard decision) related to 2 double binary code symbols per iteration (i.e. 3.5 clock cycles per symbol per iteration). With 24-byte source, there are 96 double binary coded symbols equally divided in 2 sub-blocks, i.e 48 symbol for each ASIP on each side of the turbo decoder. Hence one shuffled iteration takes around 176 cycles (3.5*48 = 168 cycles for BCJR processing + 8 cycles for state metric exchange). As far as the area of this architecture is concerned, the FPGA synthesis results of 4-ASIPs and Butterfly NoC are shown in Table.6.4.

Table 6.4 — Synthesis results of multi-ASIP and NoC based turbo decoder

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	18094 out of 207,360 (8%)
Slice LUTs	59784 out of 207,360 (28%)
DSP48Es	0 out of 192(0%)
Frequency (f)	163 MHz

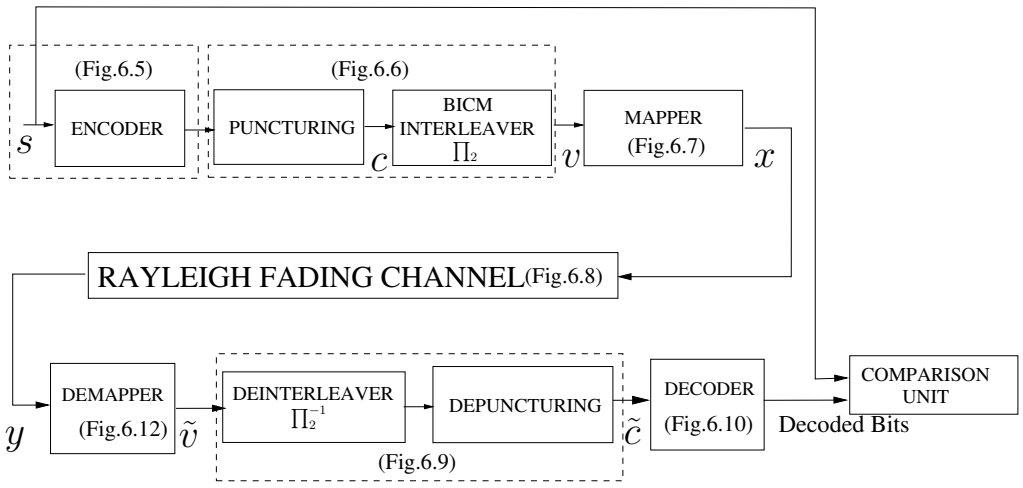


Figure 6.11 — Turbo coded transmission system implementation diagram

6.3.4 Performance Results

Using the components stated above, the complete block diagram of the communication system is shown Fig.6.11. In the implemented prototype the components of the communication system work in sequence, i.e after source generation and encoding, encoder saves data in output memories. Mapper reads output memory of encoder bit by bit, after applying $\prod_2$, to construct the modulated or mapped symbols. Each mapped symbol passes through the channel emulator and resultant symbols y and fading coefficients ρ are saved in input memories of the demapper. LLRs of each symbol y are copied into input memories of the decoders after applying $\prod_2^{-1}$ and $\prod_1$ on the fly. After the memory filling, the multi-ASIP and NoC based turbo decoder architecture execute 10 shuffled iterations to produce the decoded bits. It is worth mentioning that in first shuffled iteration, the decoders only use channel data (no extrinsic information). This is due to the fact that with a new frame under process, the extrinsic information memories hold the extrinsic data produced in the last iteration of the previous frame and thus can not be used. The comparison on the source bits and decoded bits is performed to see if the frame is correct or erroneous. The complete FPGA synthesis results of the communication system having a heterogeneous multi-ASIP platform receiver with a parallel turbo decoder are presented in Table6.5. The decoder takes 1760 cycles to decode a frame in 10 iterations. Hence, it gives a

Table 6.5 — Synthesis results of the first multi-ASIP prototype: parallel turbo decoder

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	24877 out of 207,360 (11%)
Slice LUTs	64469 out of 207,360 (31%)
DSP48Es	36 out of 192(18%)
Frequency (f)	65 MHz

throughput of 7 Mbps at a frequency of 65 MHz after 10 shuffled iterations. In fact the frequency of 65 MHz comes from the critical path of Rayleigh fading channel emulator. In practical systems one needs to implement transmitter and receiver and in this case the critical path lies in turbo decoder unit. Hence one can run the platform at 163 MHz which will result in a throughput of 17.55 Mbps after 10 shuffled decoding iterations. The acquired Frame Error Rate performance for 24-Byte source data

transmitted at $r = 0.5$ and modulated on QPSK Gray mapped constellation is shown in Fig.6.12. It has been verified that it matches the exact performance of the reference software model.

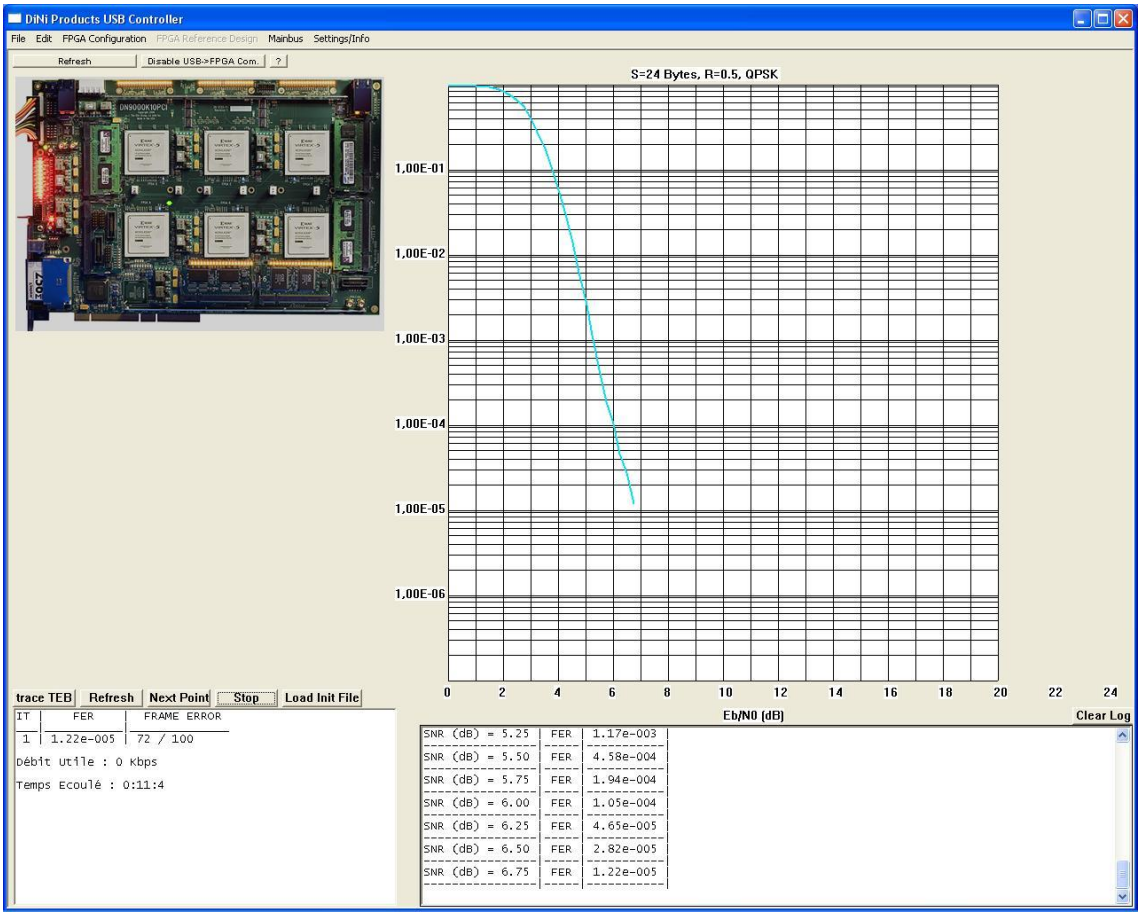


Figure 6.12 — FER performance obtained from the First multi-ASIP FPGA prototype implementing turbo decoding

6.4 Second multi-ASIP Prototype: Parallel Turbo Demodulator and Decoder

In this section we present the second multi-ASIP prototype realized to demonstrate parallel turbo demodulation and decoding. To that end, the SSD is added on the transmitter whereas on the receiver side a feed back is required in the demapper from the decoder. These all changes are shown in Fig.6.13 and will be illustrated in the following sub-sections.

6.4.1 Transmitter

On the transmitter side, to implement SSD two changes are implemented in transmitter: constellation rotation and delay d between I and Q components of symbol x. First of all contents of x_I and x_Q LUTs (Fig.6.7) are changed for a QPSK constellation rotated at angle $\alpha = 22.5°$ and last bit of the parameters (MAP STYLE) of soft mapper is set for non-Gray configuration. To implement $d = 1$,

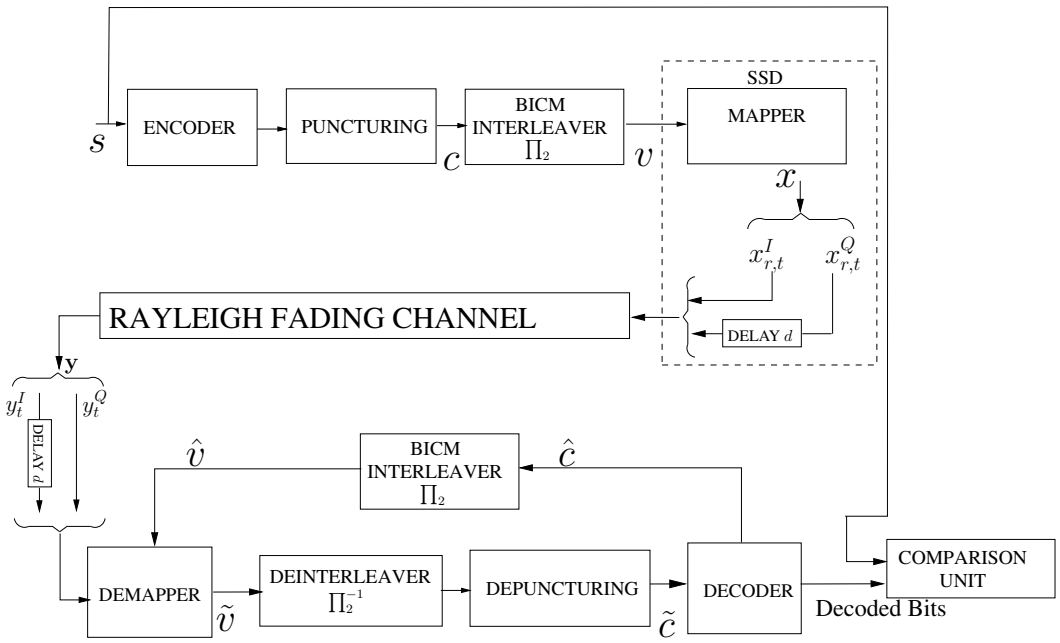


Figure 6.13 — Turbo coded with SSD transmission system diagram

193 symbols are transmitted in place of 192 symbol where the last symbol is dummy. A delay register is added for x_Q, which makes $x_Q = 0$ for the first symbol and $x_I = 0$ for the last symbol. These symbols pass through the channel and a total of 193 y and ρ are saved in demapper input memories.

6.4.2 Receiver

Multiple changes are made on the receiver side and will be detailed in this sub-section.

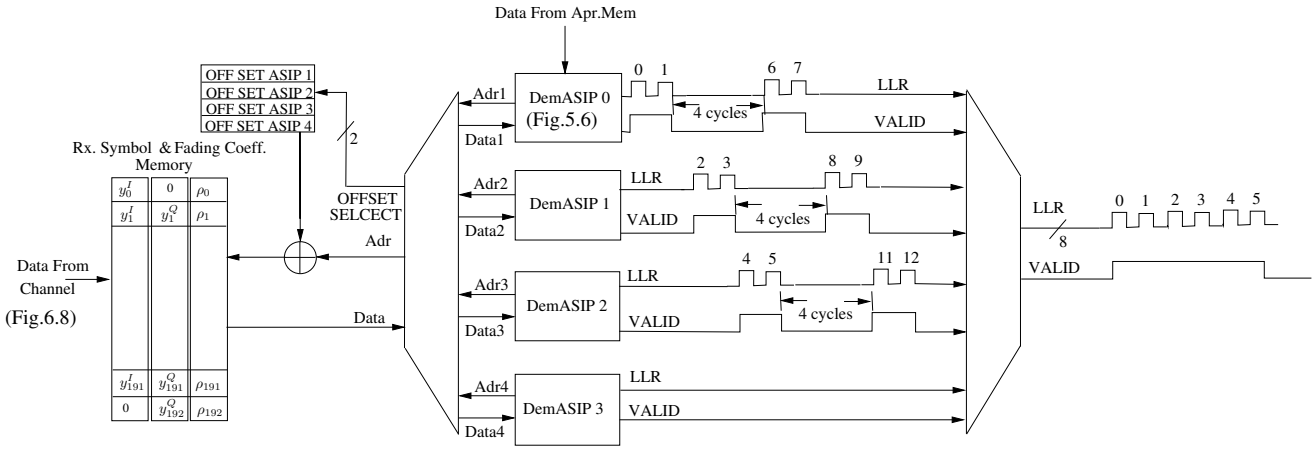


Figure 6.14 — Parallel soft demapper - 3 DemASIPs generating one LLR per clock cycle to demap QPSK symbol

6.4.2.1 Multi-ASIP Architecture for Parallel soft demapping

Two changes are made in the demapper part of the receiver. First of all, the configuration of the DemASIP is changed to implement (2.34) expression for turbo demodulation context. Due to the implementation of SSD in transmitter, the data in Rx. Symbol and Fading coeff. Memories will be in an order shown in Fig.6.14. To input fading coefficient, one needs to read ρ_0 (as ρ_{t-d}) and ρ_1 (as ρ_t) for first symbol $y_0(y^I_{t-d}, y^Q_t)$. This is implemented by first reading ρ value at 0 address of Fading Coeff. and saving it in ρ_t Register of DemASIP (Fig.5.6). Then with each input instruction, to demap an a^{th} symbol y_a, contents of ρ_t Register goes to ρ_{t-d} Register, ρ_t Register is filled by reading ρ value at address $a+1$ from Fading Coeff. Memory. Finally y^I_{t-d} and y^Q_t Registers are filled by reading two banks of Rx. Symbol Memory at location a and $a+1$ respectively.

The other issue is the processing complexity related to the used demapping expression (2.34). To implement this expression for QPSK, DemASIP takes 6 clock cycles to generate two LLRs as given in Table5.2. With non-turbo demodulation context, demapping is performed once and takes 768 cycles using one demapper and decoding takes 1408 cycles for 8 iterations (176 cycles per iteration) with 4 TurbASIPs. Hence to use DemASIP for complex expression and perform multiple demapping iterations in parallel to shuffled turbo decoding iterations, more DemASIP are required to implement shuffled turbo demodulation. One of the simple way is that if three ASIPs are used in parallel on different received symbols, implementing frame sub-blocking, one can achieve one LLR per clock cycle. Similarly with 16-QAM one need 18 clock cycles to generate 4 LLRs. Hence four ASIP will produce 16 LLRS in 18 clock cycles. The issue related to this scheme is that if all ASIPs start simultaneously, they will access the input data memories together and will produce output LLRs together.

In our case we have used a strategy which is shown in Fig.6.14. We have used 4 DemASIPs and each DemASIP starts with a delay of two clock cycles. This is achieved by adding as many NOPs in the start of the assembly code as much clock cycle delay is required. In this way the ASIPs access the input memory at different instants of time with an offset representing start address of each sub-block. At the output, with three ASIPs (the last one is switched off for the considered QPSK configuration), the throughput for QPSK will be one LLR per clock cycle whereas for 16-QAM with 4 ASIPs it will be 16 LLRs per 18 clock cycles (0.89 LLR/cycle). Hence, using three DemASIPs to demap block of 192 QPSK symbols, 384 clock cycles per iteration will be required with no memory conflict at decoder input memories. The combined synthesis results of the architecture presented in Fig.6.14 are provided in Table 6.6. With achievable frequency of 171 MHz the throughput for implemented case

Table 6.6 — Synthesis results of the multi-ASIP architecture for parallel soft demapping

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	6801 out of 207,360 (3%)
Slice LUTs	11244 out of 207,360 (5%)
DSP48Es	24 out of 192(12%)
Frequency (f)	171 MHz

of QPSK is 171 MLLR/sec and for 16-QAM it will be 152 MLLR/sec.

6.4.2.2 Modified TurbASIP Architecture

The TurbASIP used in turbo decoding application is capable of generating extrinsic information to be used inside the turbo decoder. To implement concepts of turbo demodulation and turbo equalization,

extrinsic and *a posteriori* information on bits is required. To achieve this, the max. operators available in TurbASIP are reutilized to perform the maximum operations to generate LLRs for individual systematic and parity bits. New instructions, related to these operations are added to the existing TurbASIP instruction-set. Using these instructions, the assembly program is modified. With this modified application program, 11 cycles are required to generate extrinsic related to the systematic bits of two coded symbol for turbo decoders and extrinsic related to individual systematic and parity bits for the demapper. With this new application program and for 24-byte source having 96 double binary coded symbols equally divided in 2 TurbASIPs, i.e 48 for each TurbASIP on each side of turbo decoder, one iteration takes around 272 cycles (5.5*48=264 cycles for BCJR processing + 8 cycles for state metric exchange). The FPGA implementation results of 4 TurbASIPs and Butterfly NoC are shown in Table.6.7.

Table 6.7 — Synthesis results of 4 modified TurbASIP and NoC based turbo decoder

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	19709 out of 207,360 (9%)
Slice LUTs	63624 out of 207,360 (30%)
DSP48Es	0 out of 192(0%)
Frequency (f)	135 MHz

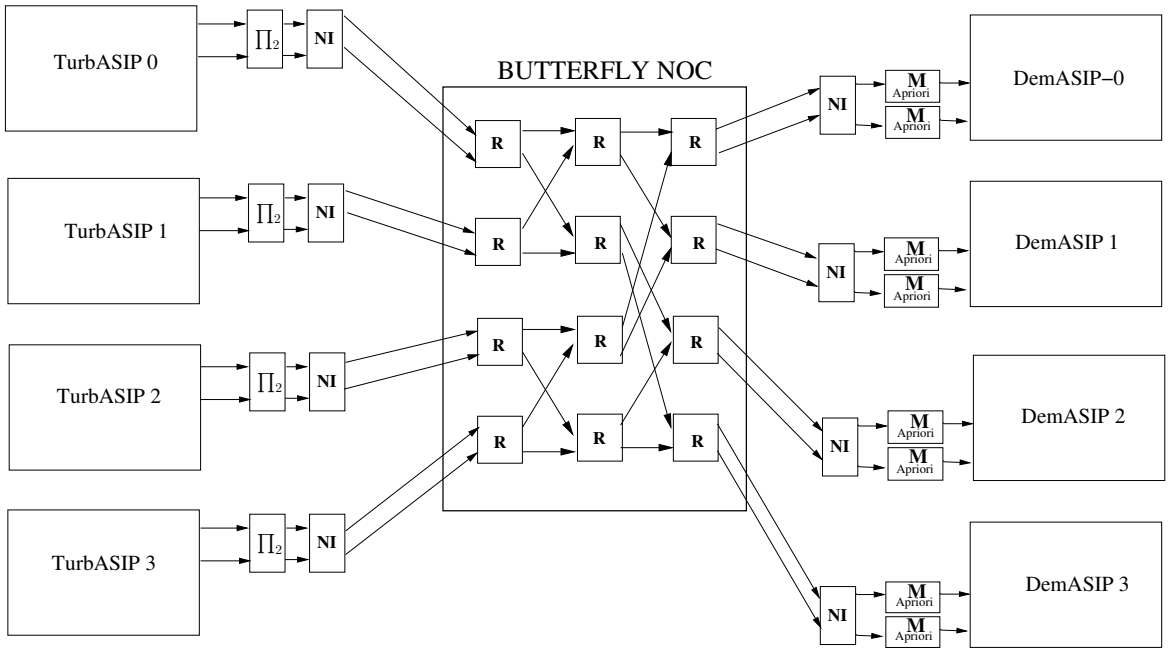


Figure 6.15 — Unidirectional Butterfly NoC between 4 TurbASIP and 4 DemASIP

6.4.2.3 Communication Network Between TurbASIPs and DemASIPs

As stated above, the TurbASIP produce LLRs related to systematic and parity bits. For a code rate $r = 0.5$, LLRs related to source bits (So,S1) and parity bit (1P0) generated by TurbASIP 0 and

TurbASIP 1 and parity bit (2P0) generated by TurbASIP 2 and TurbASIP 3 are sent to the demapper. To achieve this, a header is added to each LLR. This header contains the interleaving information related to perform $\prod_2$ interleaving in feed back path from decoder to demapper. A one way butterfly network is added for the transportation of LLRs from decoder to demapper as shown in Fig.6.15. On the demapper side each DemASIP has its own "Apriori Input Memory" which is divided into two banks. One bank stores the LLRs with even $\prod_2$ index whereas the other store the odd index. Hence "Apriori Input Memory" related to each DemASIP has two input data ports, each capable of storing one LLR, whereas the read port enables the DemASIP to read two LLRs during INPUT instruction. The synthesis results of the required communication network are summarized in Table 6.8.

Table 6.8 — Synthesis results of the unidirectional Butterfly NoC between decoder and demapper

FPGA Synthesis Results (Xilinx Virtex5 xc5vlx330)	
Slice Registers	1860 out of 207,360 (<1%)
Slice LUTs	1289 out of 207,360 (<1%)
DSP48Es	0 out of 192(0%)
Frequency (f)	265 MHz

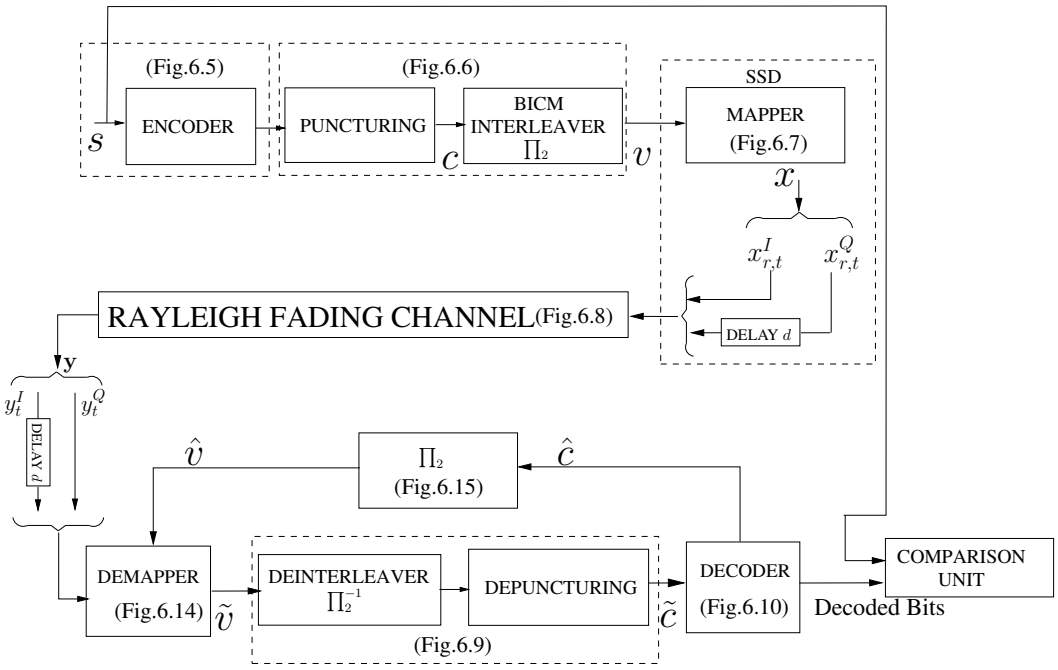


Figure 6.16 — Turbo Coded with SSD Transmission Implementation Diagram

6.4.3 Performance Results

With the proposed changes the new implementation with shuffled turbo demodulation is shown in Fig.6.16. After the serial execution of transmitter components and transmission of data from the channel, data is stored in the input memories of of the DemASIPs. The DemASIPs generates LLRs which after deinterleaving ($\prod_2^{-1}$) and depuncturing with ($\prod_1$) are saved to the input memories of

parallel turbo decoder. After this step, DemASIPs stop and TurbASIPs start working for 10 shuffled iterations where during the first iteration, the decoders only use the LLRs generated by demapper. Once "Apriori Info. Memories " of DemASIPs are filled after the first shuffled iteration of turbo decoding, DemASIPs start working again in turbo demodulation context. The reason of stopping DemASIPs during first shuffled iteration of turbo decoding is again that, during this time "Apriori Info. Memories " contain data related to the last shuffled demodulation iteration of previous processed frame.

The complete FPGA synthesis results of the communication system integrating a heterogeneous multi-ASIP platform receiver implementing turbo demodulation and decoding are shown in Table6.9. As stated above, the parallel DemASIPs architecture takes 384 clock cycles to process the frame

Table 6.9 — Synthesis results of the second multi-ASIP prototype: parallel turbo demodulator and decoder

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	33525 out of 207,360 (16%)
Slice LUTs	78016 out of 207,360 (37%)
DSP48Es	54 out of 192(28%)
Frequency (f)	65 MHz

whereas the parallel TurbASIPs architecture takes 272 cycles for the same frame. Hence during shuffled turbo demodulation process there will be 9 shuffled turbo decoding iterations and almost 6.3 shuffled turbo demodulation iterations. The total number of cycles required to process a frame of 24 byte source data will be 3104 ($384 + 272 \times 10$) which will result in a throughput of 4 MBits/sec at 65 MHz clock frequency. But again this 65 MHz is due to channel's critical path. However, the critical path of this receiver lies in the modified TurbASIP architecture and enables a maximum clock frequency of 135 MHz. With this frequency the throughput of the receiver will be 8.3 MBits/sec. The acquired Frame Error Rate performance for 24-Byte source data transmission at $r = 0.5$ and modulated on QPSK rotated constellation with SSD is shown in Fig.6.17. The results shown a gain of 0.5 dB with the inclusion of SSD in transmitter and turbo demodulation in receiver as compared to the FER results of turbo decoding presented in Fig.6.12.

6.5 Third multi-ASIP Prototype: Parallel Unified Turbo Receiver

In this section we present the first multi-ASIP prototype realized to demonstrate parallel turbo decoding.

By adding the concept of turbo equalization to the second multi-ASIP prototype presented above, we conclude with this third heterogeneous multi-ASIP prototype which can implement all three concepts of turbo decoding, demodulation and equalization. The complete block diagram is shown in Fig.6.18. STC of 2×2 MIMO SM is added on the transmitter side whereas MIMO MMSE-IC ASIP equalizer and soft mapper is added in the receiver.

6.5.1 Transmitter

On transmitter side, SSD is removed and 2×2 MIMO SM is added. It is done in a way that when two symbols are produced by the mapper, the two symbols with a flag of vector ready is given at the

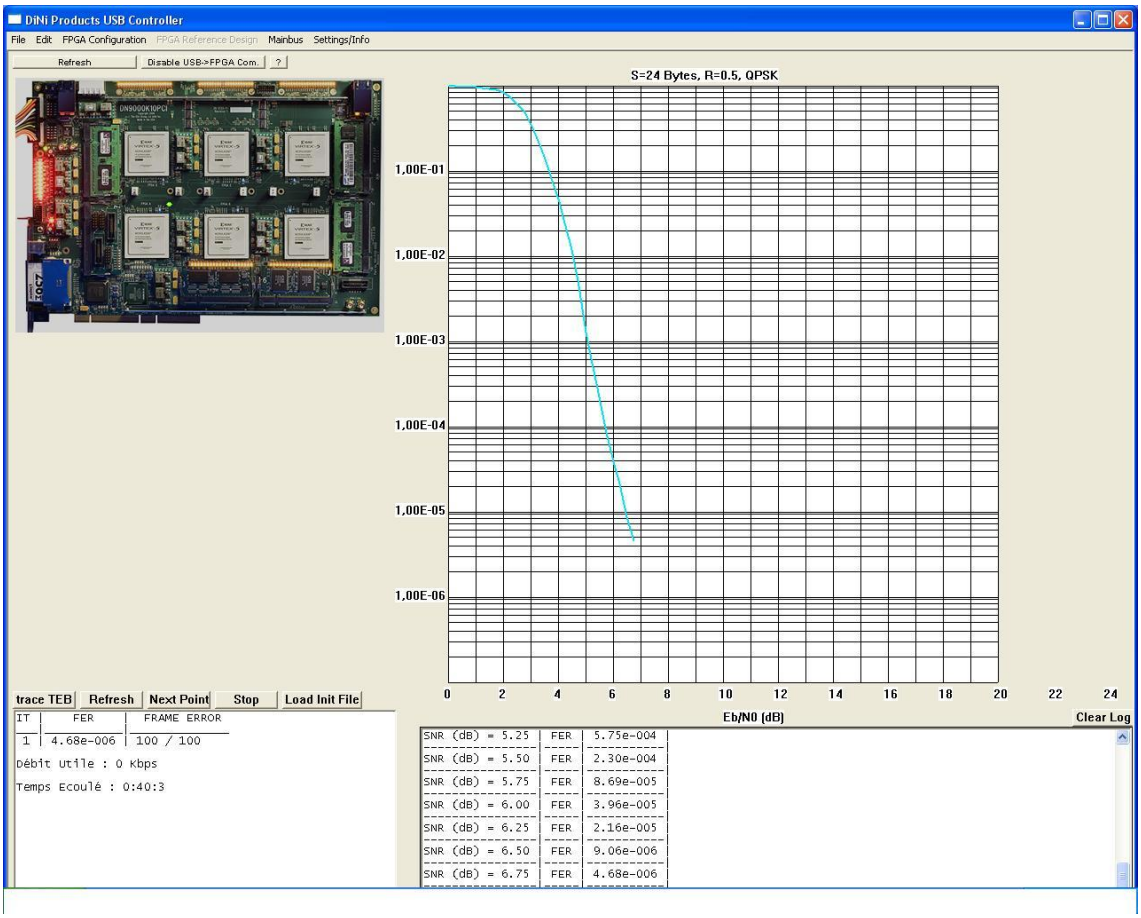


Figure 6.17 — FER Obtained from the second multi-ASIP Prototype implementing turbo demodulation and decoding

output of the transmitter.

6.5.2 MIMO Flat Block Rayleigh Fading Channel

For the prototyping of the MIMO system, the channel which is used in the previous two systems is replaced with a MIMO channel. The inputs and outputs of this channel emulator is shown in Fig.6.19. The channel model is configured for flat block Rayleigh Fading Channel, in which channel remains constant for two transmitted vectors. The input to the channel are two symbols $x1$ and $x2$ having I and Q components each composed of 8 bits. H_REFRESH is toggled after each two received vectors to have new values of channel matrix. The rest of the inputs i.e. σ_w^2, CLK, RESET and ENABLE are same as described in previous channel model. The outputs on the channel are $y1$ and $y2$ having I and Q components each having 12 bits. Four elements of channel matrix $\mathbf{H}$ are represented with $h11$, $h12$, $h21$ and $h22$. The individual components of $\mathbf{H}$ is composed of I and Q components each having 12 bits. The output ready gives an indication to save the channel output in the input memories of the equalizer. The FPGA resources of the channel emulator are summarized in Table 6.10.

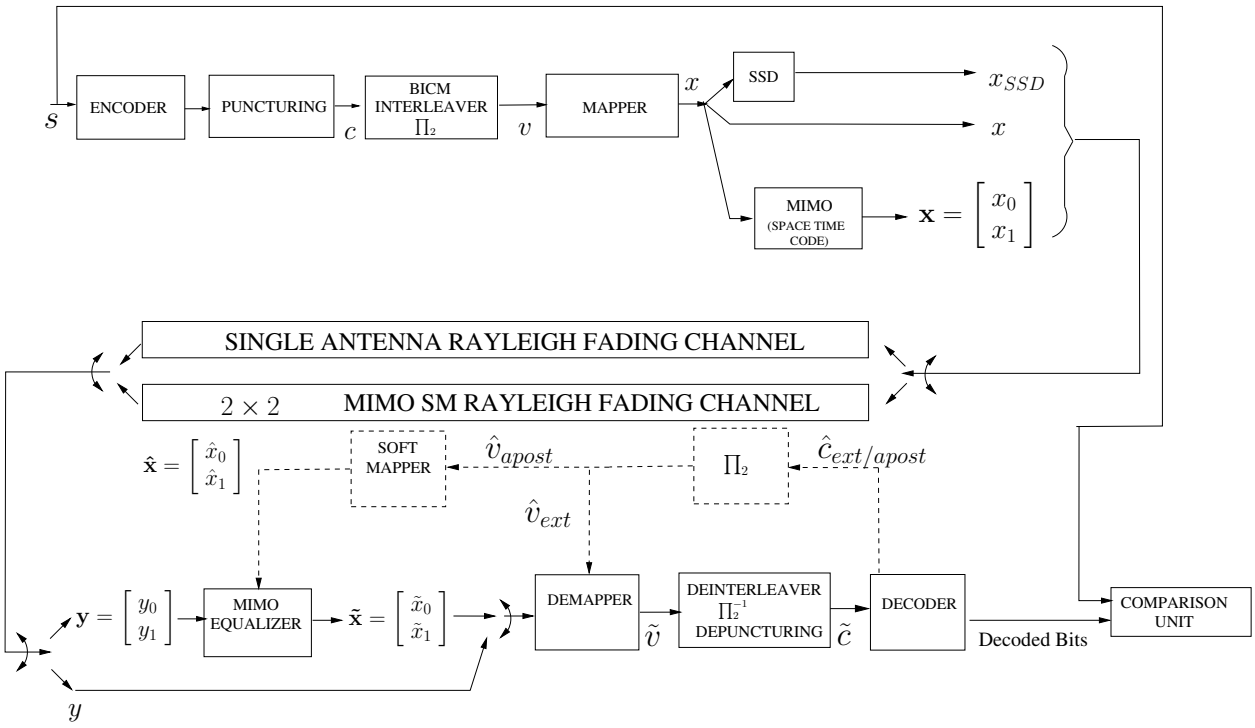


Figure 6.18 — Turbo Coded with MIMO STC Transmission Diagram

Figure 6.19 — MIMO Rayleigh fading channel emulator

6.5.3 Receiver

On the receiver side, a MIMO equalizer and a soft mapper (Fig.6.18) is added to complete the path for all possible loops in a complete turbo receiver.

6.5.3.1 MIMO equalizer

To perform MIMO equalization, one EquASIP of Fig.4.6 is integrated in the receiver. The application program for this ASIP is written for 2×2 MIMO SM. For the application of a block fading channel where channel is constant for two vectors, the throughput of the EquASIP is two estimated vectors ($\tilde{\mathbf{x}}$) and bias vector ($\mathbf{g}$) i.e 4 $\tilde{x}$ symbols and associated bias g per 80 clock cycles. These output go in the demapper for symbol to LLR conversion. A change is made in the contents of inv. Sigma LUT of DemASIP (Fig. 5.6). This change is related to the explanation presented in subsection 2.3.2 where σ_w^2

Table 6.10 — Synthesis results of MIMO Rayleigh fading channel block

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	6036 out of 207,360 (2%)
Slice LUTs	6203 out of 207,360 (2%)
DSP48Es	96 out of 192(50%)
Frequency (f)	65 MHz

is replaced with $g(1-g)\sigma_x^2$. Hence in MIMO case, the address line of LUT is g (not σ_w^2) and contents of inv. Sigma LUT represent $g(1-g)\sigma_x^2$. The TurbASIP's program is also modified to generate *a posteriori* LLRs for the soft mapper. During first iteration of equalizer the $\hat{x}$ and $\sigma_{\hat{x}}^2$ are zero whereas MMSE-IC2 is implemented for the sake of simplicity where $\sigma_{\hat{x}}^2 = \sigma_x^2$.

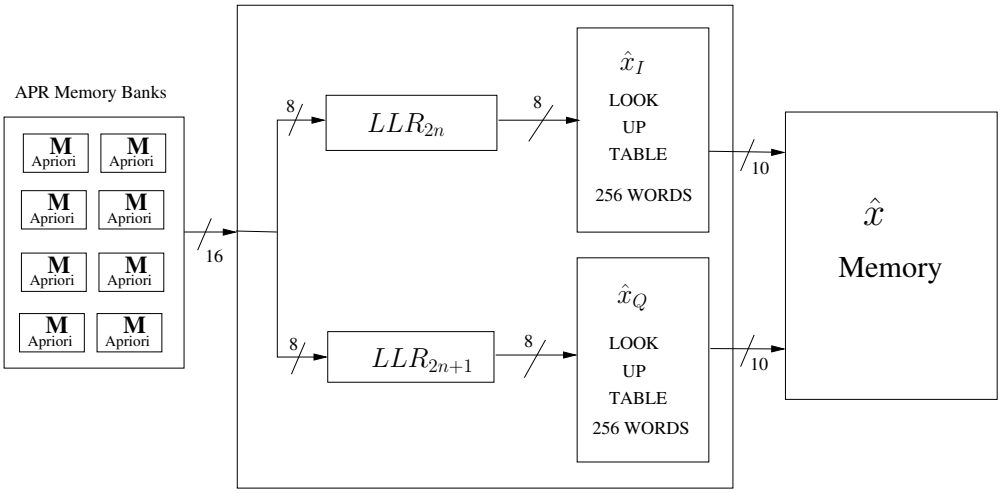


Figure 6.20 — Soft mapper for QPSK Configuration

6.5.3.2 Soft Mapper

For the demonstration purpose a dedicated architecture of QPSK soft mapper is written in VHDL. The architecture is simple as shown in Fig.6.20. The input to soft mapper comes from the APR memory banks of Fig.6.15. The soft mapper reads 16 bits which contain two *a posteriori* LLRs of 8 bits. The LLR at even numbered address (LLR_{2n}) constructs $\hat{x}_I$ while the LLR at odd numbered address (LLR_{2n+1}) is involved in computing $\hat{x}_Q$. The both look up tables contain the values related to following expression for soft mapping.

$$\hat{x}_n(I) = -tanh\frac{LLR_{2n}}{2} \text{ and } \hat{x}_n(Q) = -tanh\frac{LLR_{2n+1}}{2}$$

Using input LLRs, the components of $\hat{x}$ are read from LUT and are saved in the $\hat{x}$ Memory which is read by EquASIP.

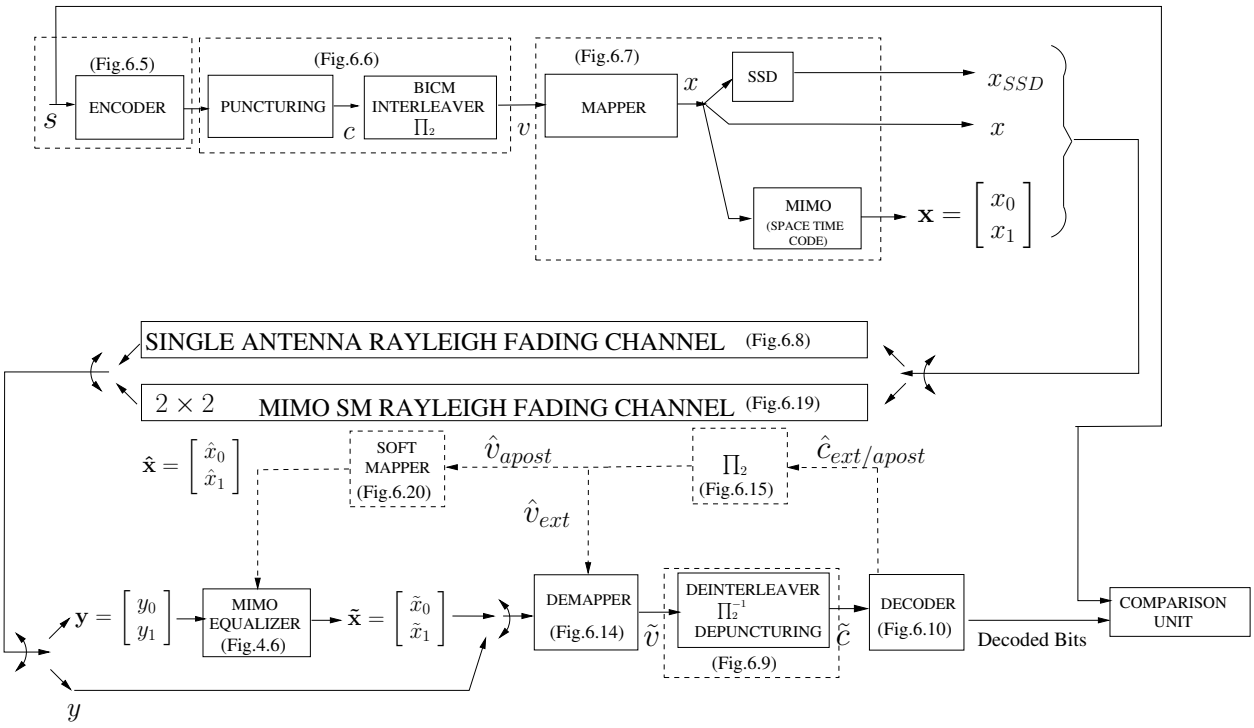


Figure 6.21 — Turbo coded with MIMO STC transmission diagram

6.5.4 Performance Results

With the proposed changes the new implementation with shuffled turbo equalization is shown in Fig.6.21. After the serial execution of transmitter components and transmission of data from the channel, data is stored in the input memories of EquASIPs. The EquASIP produces estimated symbols $\tilde{x}$. One of the demapper in DemASIPs architecture generates LLRs which after deinterleaving ($\prod_2^{-1}$) and depuncturing with ($\prod_1$) are saved in input memories of the turbo decoder. After this step EquASIP and DemASIPs architecture stops and TurbASIPs architecture works for 10 shuffled iterations where during first iteration decoders only use LLRs generated by demapper. Once "Apriori Info. Memories " is filled after the first shuffled iteration of turbo decoding function, soft mapper produces decoded symbols $\hat{x}$. Once all soft symbols are produced, soft mapper, EquASIP and DemASIPs architecture starts to work again in turbo equalization context and hence system implements shuffled turbo equalization. The reason of stopping soft mapper, EquASIP and DemASIPs architecture during first shuffled iteration of turbo decoding architecture is again that, during this time "Apriori Info. Memories " contain data related to last shuffled equalization iteration of previous processed frame.

The complete FPGA synthesis results of the communication system integrating a heterogeneous multi-ASIP platform receiver implementing turbo equalization, demodulation and decoding are shown in Table6.11. As stated above, one EquASIP takes 80 cycles to process two received vectors (on average 40 cycles per vector). With a 24 bytes source transmitted at $r = 0.5$ using QPSK with 2×2 MIMO SM, there are 96 vectors to be processed. Hence, the total time for equalization process is 3840. Since demapping can work in pipeline fashion, demapping runs in pipeline with equalization. As stated before the turbo decoder architecture consumes 272 cycles per shuffled iteration (2720 cycles for 10 shuffled iterations). Hence during shuffled turbo equalization process there will be less than one shuffled turbo equalization iteration. This provides no significant gain in turbo equalization process.

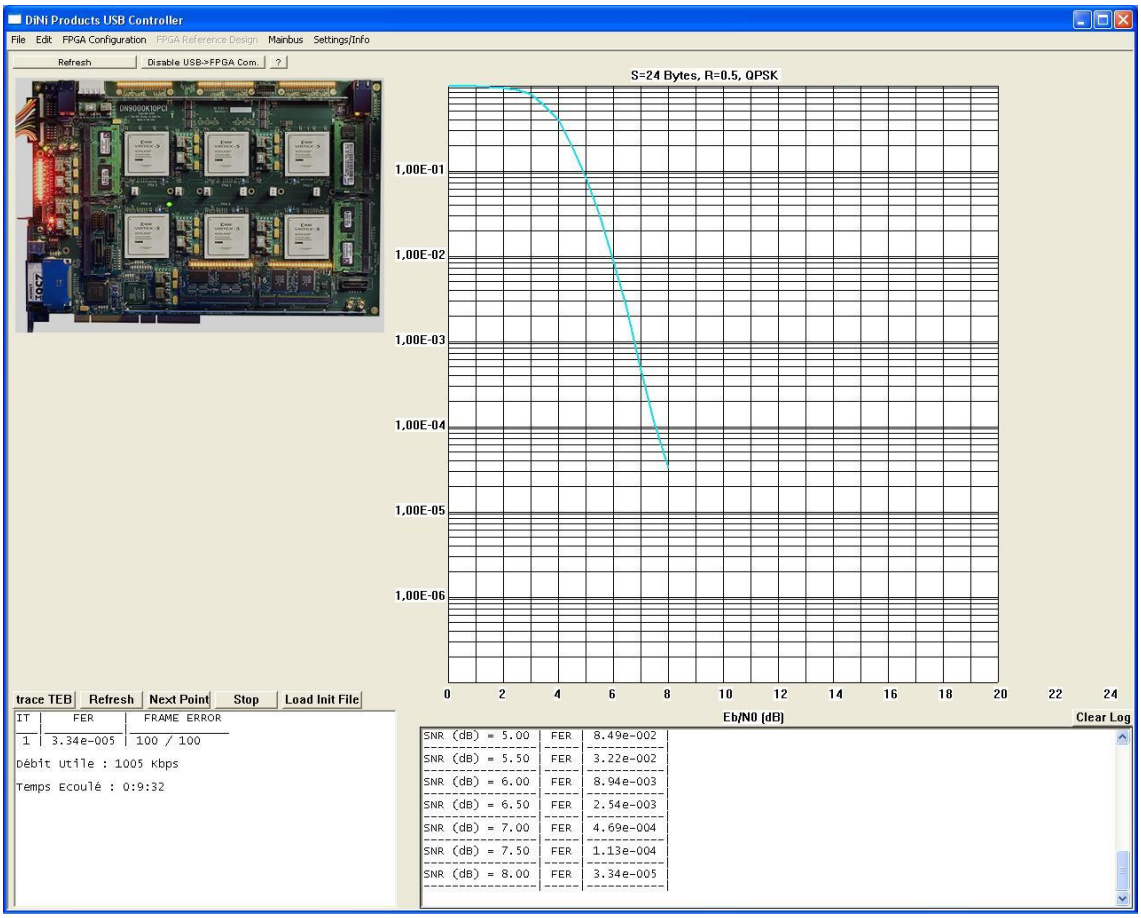


Figure 6.22 — FER Obtained from Third multi-ASIP Prototype implementing the unified turbo receiver

Table 6.11 — Synthesis Results of the third multi-ASIP prototype: parallel turbo equalizer, demodulator and decoder

FPGA Synthesis Results(Xilinx Virtex5 xc5vlx330)	
Slice Registers	36751 out of 207,360 (17%)
Slice LUTs	88387 out of 207,360 (42%)
DSP48Es	132 out of 192(68%)
Frequency (f)	65 MHz

The total number of cycles required to process a frame of 24 byte source data will be 6560 ($3840 + 2720$) which will result in a throughput of 1.9 MBits/sec at 65 MHz clock frequency. But again this 65 MHz is due to channel's critical path. However, if we only consider the critical path of the receiver, it lies in EquASIP i.e 130 MHz. With this frequency the throughput of the receiver will be 3.8 MBits/sec. The acquired Frame Error Rate performance for 24-Byte source data transmitted at $r = 0.5$, using QPSK modulation and 2×2 MIMO SM is shown in Fig.6.22. Since sub-block parallelism in equalization is simple hence, with the addition of three more EquASIPs the first equalization time will reduce to 960 cycles and hence total cycles required will be 3680. This will result in a throughput of 6.75 MBits/sec and more shuffled turbo equalization iterations.

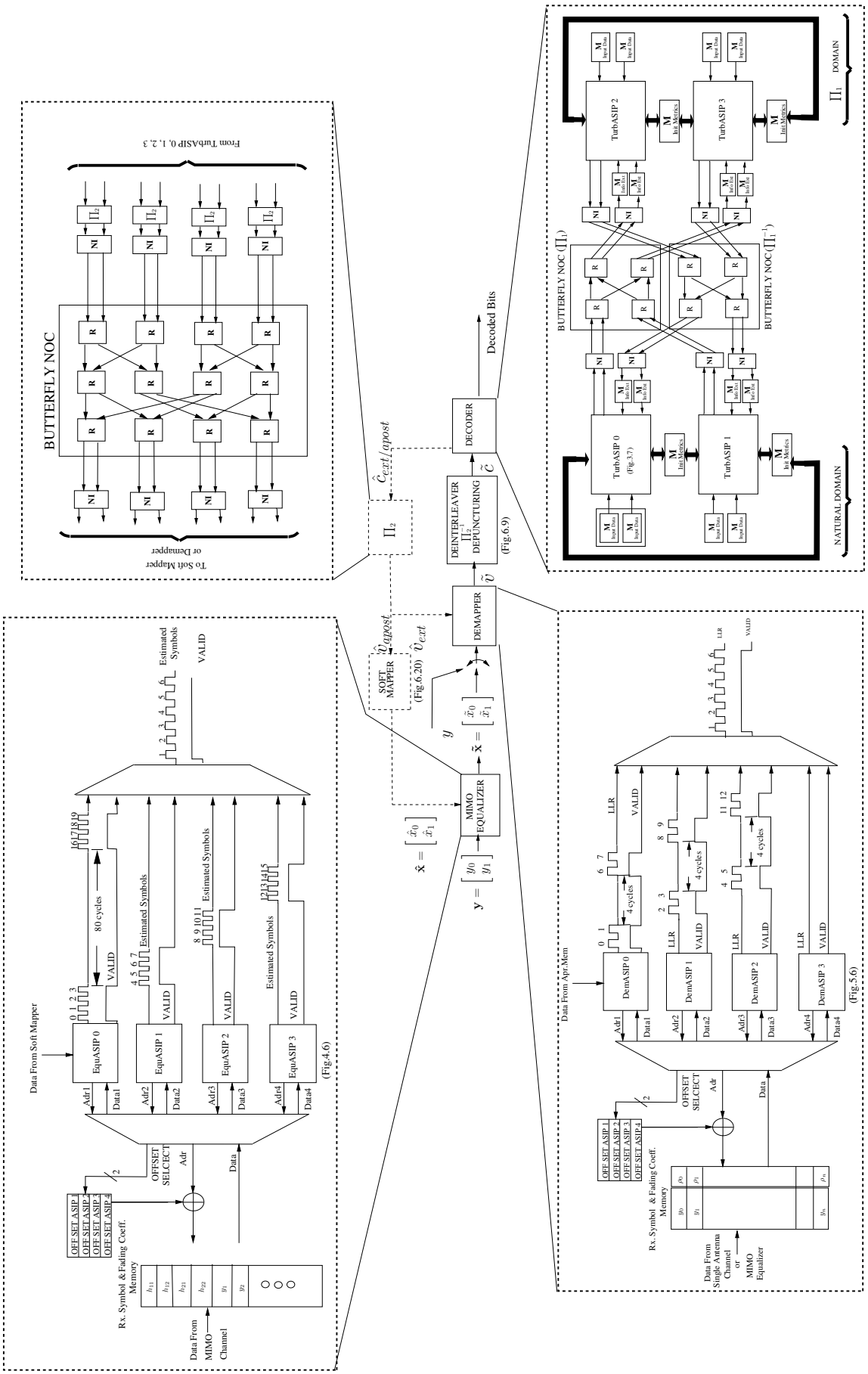


Figure 6.23 — Unified turbo receiver's detailed diagram

With more EquASIPs one can see the effect of more equalization iteration in the shape of FER performance improvement. The detailed diagram of unified turbo receiver with 4 instances of each of EquASIP, DemASIP and TurbASIP with 3 Butterfly based NoCs is shown in Fig.6.23.

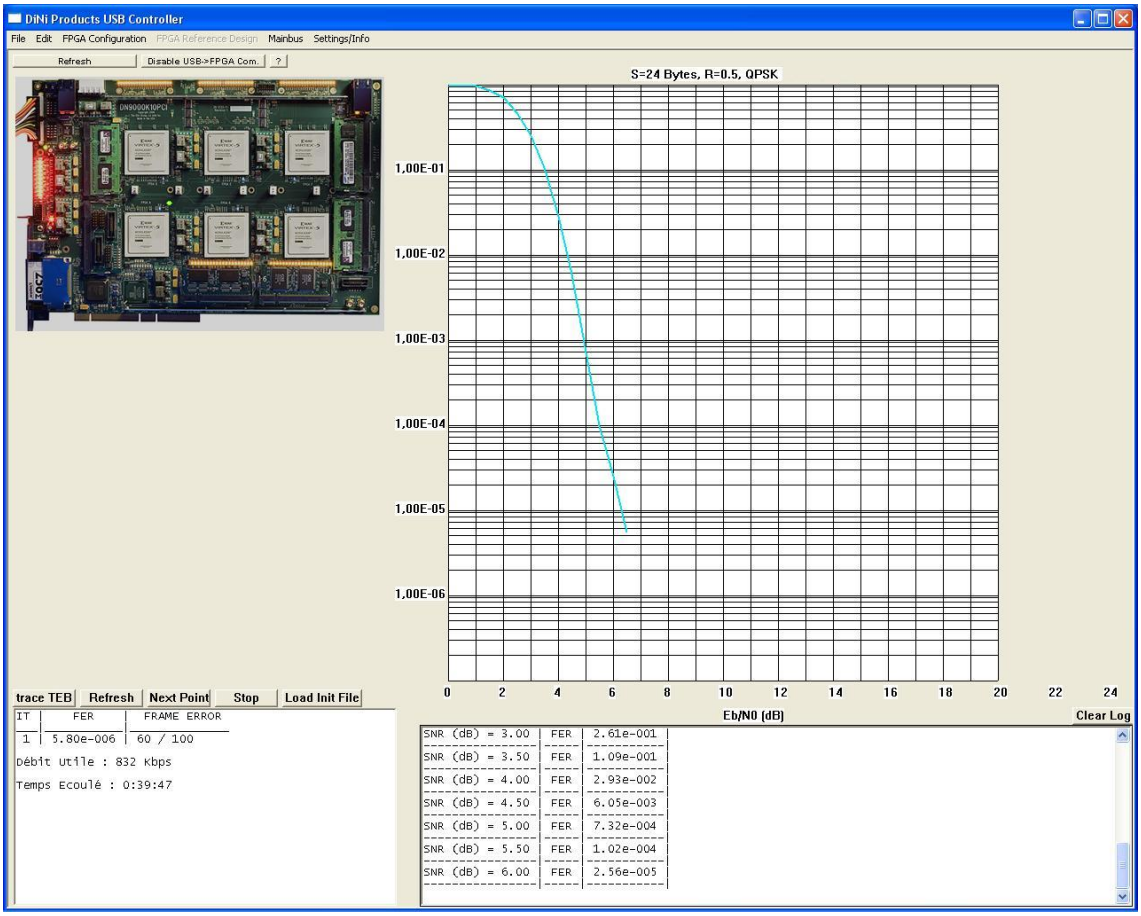


Figure 6.24 — FER Obtained from Third multi-ASIP FPGA Prototype Implementing Shuffled Turbo Equalization with Perfect *a priori* to Equalizer

Due to shortage of time, required for the integration of more EquASIPs, an indirect approach is used to validate the platform. In the adopted approach the perfect information is provided to the equalizer and in Fig.6.24 one can see a gain of almost 1.5 dB which validates the EquASIP in iterative equalization context.

6.6 Conclusion

In this chapter, original heterogeneous multi-ASIP prototypes are proposed towards the design of a unified iterative receiver. These prototypes demonstrate the efficient exploitation of the second level of parallelism available in turbo equalization, demodulation and decoding applications, introduced in chapter 2 as Component Level Parallelism. The first level of parallelism (Metric or Symbol Estimation Level Parallelism) has been efficiently exploited inside each one of the individual proposed ASIPs as explained in Chapters 4 and 5.

The chapter has started by illustrating the design, validation and prototyping flow for individual ASIPs. These ASIP are initially described in LISA ADL and then went through the processes of

VHDL generation and FPGA validation. Once having these ASIPs validated on FPGA, a step by step integration process is described. The system started with shuffled turbo decoding through the first multi-ASIP prototype. SSD on transmitter and turbo demodulation on receiver side is then demonstrated by means of a second multi-ASIP prototype. With the implementation of of turbo demodulation flexibility of DemASIP is demonstrated by using it in iterative context. In addition, the NoC solution is adopted to feedback the interleaved information from decoder to demapper. Finally a third heterogeneous multi-ASIP architecture is presented as a unified iterative receiver which integrates parallel turbo demodulation, equalization and decoding. These final FPGA prototype implements flexible transmitter, channel emulator and a turbo receiver made up of 9 ASIP of three different types and 3 NoC instances using Butterfly topology. With each prototype the results of synthesis and FER results, for different system configurations, acquired from FPGA platform are presented and verified to match the exact performance of the corresponding reference software model.

Conclusion and perspectives

THE work accomplished in this PhD thesis considers flexibility and throughput requirements of future wireless standards in a context where turbo processing is applied in the receiver to attain error rate performance close to theoretical limits.

Firstly, throughput and flexibility parameters associated to different functional blocks of the transmitter have been extracted from multiple emerging wireless communication standards. In the presence of techniques such as turbo codes, SSD and MIMO on the transmitter side, there is an encouragement for iterative processing implementation in the receiver to target error rate performance issue. Hence, at the end of chapter 1, a scheme for a unified turbo receiver is proposed.

As high throughput and serial iterative processing are contrary to each other, a parallelism study is conducted on turbo demodulation and turbo equalization at different levels by inheriting the available results of parallel turbo decoding. Based on the study about the first two parallelism levels (metric/symbol generation level and component level), simulation results are presented to quantify the effects of parallelism implementation.

While targeting efficient hardware implementation, the ASIP approach is adopted to exploit the first parallelism level and to achieve the required tradeoff between flexibility and performance. In this context two original Soft-In Soft-Out ASIP architectures have been proposed and designed, namely EquASIP and DemASIP. EquASIP has been presented as the first flexible ASIP implementing an MMSE-IC linear equalizer for turbo equalization application. Analysis and simulation of mathematical equations involved in MMSE-IC LE allowed to identify potential complex-numbered operations which lead to device the instruction set. The specific instructions for complex number arithmetic enable to efficiently perform computations on variable sized complex numbered matrices which in turn provide required flexibility in MMSE-IC and promote its reuse for other MMSE-based applications. EquASIP flexibility allows its reuse for each of Alamouti code, Golden code, 2×2, 3×3 or 4×4 spatially multiplexed turbo MIMO application with BPSK, QPSK, 16-QAM, and 64-QAM. When targeting 90 nm technology, the proposed architecture enables a maximum throughput of 273 MSymbol/sec for 2×2, 148 MSymbol/sec for 3×3 and 168 MSymbol/sec for 4×4 MIMO systems.

The second proposed ASIP architecture, DemASIP, constitutes the first flexible ASIP implementing a universal demapper for multi wireless communications standards. Following a first step of LLR computational expressions analysis and identification of flexibility parameters and common operators, a specialized instruction set has been designed. The architecture proposed addresses all the complexity levels associated with demapping functionality through the arrangement of certain LUTs and the Euclidean Unit which allows its reuse both in an iterative and a non-iterative context and provides support for BPSK to 256-QAM constellation for any mapping style used. The specialized instruction set provides ability to generate the required LLRs for different system configurations. When targeting 90 nm technology, the proposed architecture enables a maximum throughput of 358 Mega LLRs per second for 16-QAM Gray mapped constellation.

The two proposed ASIPs have been also validated by means of an FPGA prototype using a logic emulation board (DN9000K10PCI) integrating Xilinx Virtex 5 devices. The adopted design, validation and prototyping flow, using Processor Designer Framework from CoWare Inc., has been detailed.

These ASIPs are initially described in LISA ADL and then went through the processes of VHDL generation and on-board FPGA validation.

Exploiting the second level of parallelism (Component Level Parallelism) in turbo equalization, demodulation and decoding is then done through the proposal of original heterogeneous multi-ASIP prototypes towards the design of a unified iterative receiver. To that end, three incremental complexity multi-ASIP prototypes have been realized. The first one demonstrates parallel turbo decoding, the second demonstrates parallel turbo demodulation and decoding and the third one demonstrates parallel turbo demodulation, equalization and decoding. The proposed multi-ASIP architectures demonstrate the efficient exploitation of sub-blocking and shuffled techniques of the second level of parallelism. For all of these three multi-ASIP prototypes, hardware implementation of functional blocks of transmitter, channel and receiver were described. The final (third) FPGA prototype implements flexible transmitter, channel emulator and a unified turbo receiver made up of 9 ASIPs of three different types and 3 NoC instances using Butterfly topology. For all realized prototypes, the results of synthesis and FER results, for different system configurations, acquired from FPGA platform were also presented and verified to match the exact performance of the corresponding reference software model.

The presented original contributions demonstrate promising results using a heterogeneous multi-ASIP and NoC based approach to implement flexible, yet efficient, unified iterative receiver.

Perspective

Regarding work perspectives, for the short term, as memory look up tables are used for interleaving/deinterleaving functions, it would be interesting to investigate their replacement with a flexible hardware implementing the mathematical expression for interleaving/deinterleaving functions. This will give two benefits in the shape of reduced memory use and relative high reconfiguration speed.

For the long/mean term work perspective, the low-power requirement should be considered as another optimizing design dimension to the multi-ASIP NoC solution for wireless communications applications. As in the second level parallelism implementation, the required processing power can be achieved by switching on the required number of ASIPs, the proposed architecture provides a natural power management scheme at this level. However, inside each ASIP a room is still available to apply power reduction techniques. By inheriting the relevant low-power design techniques from already established low-power implementation schemes for programmable architectures, the conceived ASIPs can be optimized for power consumption. Furthermore, in this thesis work an attempt is made to provide maximum flexibility within the two designed ASIPs for demapping and equalization. This upper bound for flexibility implies similar bound for power consumption. Hence by examining the exact required flexibility for a particular system a tradeoff can be achieved between flexibility and power demands.

Since a unified architecture of turbo receiver is now available, the other possible direction is the study of overall optimality of the architecture under different channel conditions. This includes the study of stopping criteria of different processing units according to considered channel conditions to reach the required error rate performance. With the information of such criteria, the receiver can be further optimized for energy consumption.

Finally, other flexible functional blocks can be conceived to complete the baseband part of the receiver. These flexible blocks are required to perform functions such as synchronization, OFDM interfacing and channel estimation.

Glossary

3GPP	3rd Generation Partnership Project
ACS	Addition Comparaison Selection
ADL	Architectural Description Language
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction-set Processor
AWGN	Additive White Gaussian Noise
BCJR	Bahl-Cock-Jelinek-Raviv
BICM	Bit Interleaved Coded Modulation
BPSK	Binary Phase Shift Keying
CAD	Computer Aided Design
CAU	Complex Arithmetic Unit
CCASM	Combined Complex Adder Subtracter Multiplier
CORDIC	Coordinate Rotation Digital Computer
DE-LST	Diagonal Encoding - Layered Space Time
DVB-RCS	Digital Video Broadcasting Return Channel Satellite
DVB-T	Digital Video Broadcasting Terrestrial
DSP	Digital Signal Processor
ECC	Error Control Coding
EU	Euclidean Unit
FER	Frame Error Rate
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FSK	Frequency Shift Keying
HDL	Hardware Description Language
HE-LST	Horizontal Encoding - Layered Space Time
ID	Iterative Demapping
ISI	Inter Symbol Interference
LDPC	Low-Density Parity-Check

LLR	Log Likelihood Ratio
LTE	Log Term Evolution
LUT	Look Up Table
MAP	Maximum A Posteriori
MIMO	Multiple Input Multiple Output
ML	Maximum Likelihood
MMSE-IC	Minimum Mean Square Error-Interference Canceller
MPSoC	Multiple Processor System on Chip
MRB	Matrix Register Bank
NI	Network Interface
NoC	Network on Chip
OCP	Open Core Protocol
OFDM	Orthogonal Frequency Division Multiplexing
PC	Pre-coding
PCCC	Parallel Concatenated Convolutional Codes
PCI	Peripheral Component Interconnect
PE	Processing Element
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfert Level
SCCC	Serially Concatenated Convolutional Codes
SD	Sphere Decoding
SGR	Standard Givens Rotation
SISD	Soft In Soft Out
SM	Spatial Multiplexing
SNR	Signal to Noise Ratio
SRD	Software Defined Radio
SoC	System on Chip
SOVA	Soft Output Viterbi Algorithm
SSD	Signal Space Diversity
STBC	Space Time Block Code
STC	Space Time Code
STTC	Space Time Trellis Code
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
VCI	Virtual Component Interface
HE-LST	Vertival Encoding - Layered Space Time

ZOL	Zero Overhead Loop

Bibliography

[1] C. Shannon, "A mathematical theory of communication," Bell System Technical Journal, Tech. Rep., 1948.

[2] G. J. Forney, *"Performance of concatenated codes", Key papers in the development of coding theory*, E. Berlekamp, Ed. IEEE Press, 1974.

[3] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *IEEE International Conference on Communications, ICC 93.*, vol. 2, 1993, pp. 1064–1070 vol.2.

[4] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," The Telecommunications and Data Acquisition Report, Tech. Rep. pp. 56-65., 1995.

[5] C. Douillard, M. Jezequel, C. Berrou, J. Tousch, N. Pham, and N. Brengarth, "The Turbo Code Standard for DVB-RCS," in *2nd International Symposium on Turbo Codes and Related Topics, Brest, France*. ELEC - Dépt. Electronique (Institut Télécom-Télécom Bretagne), 2000, pp. 535 – 538.

[6] *802.16 IEEE Standard for Local and metropolitan area networks, Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, Std., 2004.

[7] *3GPP Technical Specifications 36.212, Multiplexing and Channel Coding (Release 8)*, Std., 2008.

[8] E. Zehavi, "8-PSK trellis codes for a Rayleigh channel," *IEEE Trans. Commun.*, vol. 40, no. 5, pp. 873–884, May 1992.

[9] G. Caire, G. Taricco, and E. Biglieri, "Bit-interleaved coded modulation," *IEEE Trans. Inform. Theory*, vol. 44, no. 3, pp. 927–946, May 1998.

[10] C. Abdel Nour, "Spectrally Efficient Coded Transmission for Wireless and Satellite Applications," Ph.D. dissertation, TELECOM Bretagne, Department of Electronics, Brest, France, 2008.

[11] J. Tan and G. Stuber, "Analysis and design of symbol mappers for iteratively decoded BICM," *IEEE Trans. Wireless Commun.*, vol. 4, no. 2, pp. 662–672, Mar. 2005.

[12] S. Al-Semari and T. Fuja, "I-Q TCM: reliable communication over the Rayleigh fading channel close to the cutoff rate," *IEEE Trans. Inform. Theory*, vol. 43, no. 1, pp. 250–262, 1997.

[13] A. Chindapol and J. Ritcey, "Design, analysis, and performance evaluation for BICM-ID with square QAM constellations in Rayleigh fading channels," *IEEE J. Select. Areas Commun.*, vol. 19, no. 5, pp. 944–957, May 2001.

[14] V. Tarokh, N. Seshadri, and A. R. Calderbank, "Space-time codes for high data rate wireless communication: performance criterion and code construction," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 744–765, March 1998.

[15] S. M. Alamouti, "A simple transmit diversity technique for wireless communications," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 8, pp. 1451–1458, October 1998.

[16] V. Tarokh, H. Jafarkhani, and A. R. Calderbank, "Space-time block codes from orthogonal designs," *IEEE Transcations in Information Theory*, vol. 45, no. 5, pp. 1456–1467, July 1999.

[17] B. Hassibi and B. M. Hochwald, "High-rate codes that are linear in space and time," *IEEE Transactions on Information Theory*, vol. 48, pp. 1804–1824, 2002.

[18] J. C. Belfiore, G. Rekaya, and E. Viterbo, "The Golden code: a 2×2 full-rate space-time code with non-vanishing determinants," in *ISIT'04, IEEE International Symposium on Information Theory*, June-July 2004, pp. 310–310.

[19] G. J. Foschini, "Layered space-time architecture for wireless communication in fading environments when using multi-element antennas," *Bell Labs Technical Journal*, pp. 41–59, 1996.

[20] X. Li and J. Ritcey, "Bit-interleaved coded modulation with iterative decoding," *IEEE Commun. Lett.*, vol. 1, no. 6, pp. 169–171, Nov. 1997.

[21] I. Abramovici and S. Shamai, "On turbo encoded BICM," *Ann. Telecommun.*, vol. 54, no. 3, pp. 225–234, Mar. 1999.

[22] C. Abdel Nour and C. Douillard, "On lowering the error floor of high order turbo BICM schemes over fading channels," in *IEEE Global Telecommun. Conf., GLOBECOM*, Nov. 2006, pp. 1–5.

[23] C. Douillard, M. Jézéquel, C. Berrou, A. Picart, P. Didier, and A. Glavieux, ""Iterative correction of inter symbol interference : Turbo equalization"," *European Trans. on Telecom.*, vol. vol. 6, pp. pp. 507–511, Sept-Oct 1995.

[24] R. Fano, "A heuristic discussion of probabilistic decoding," *IEEE Transactions on Information Theory*, vol. 9, no. 2, pp. 64–74, 1963.

[25] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[26] J. Forney, G.D., "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[27] J. Hagenauer and P. Hoeher, "A viterbi algorithm with soft-decision outputs and its applications," in *IEEE Global Telecommunications Conference, 1989, and Exhibition. 'Communications Technology for the 1990s and Beyond'. GLOBECOM '89.*, 1989, pp. 1680–1686 vol.3.

[28] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate (corresp.)," *IEEE Transactions on Information Theory*, vol. 20, no. 2, pp. 284–287, 1974.

[29] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *IEEE International Conference on Communications, 1995. ICC 95 Seattle, Gateway to Globalization*, vol. 2, 1995, pp. 1009–1013 vol.2.

[30] C. Douillard and C. Berrou, "Turbo codes with rate-m/(m+1) constituent convolutional codes," *IEEE Transactions on Communications*, vol. 53, no. 10, pp. 1630–1638, 2005.

[31] C. Berrou, *Codes and Turbo Codes*. Springer, 2010.

[32] M. Tuchler, A. C. Singer, and R. Koetter, "Minimum mean squared error equalization using a priori information," *IEEE Transactions on Signal Processing*, vol. 50, no. 3, pp. 673–683, March 2002.

[33] R. Le Bidan, "Turbo-equalization for bandwidth-efficient digital communications over frequency-selective channels," Ph.D. dissertation, INSA Rennes - Institut National des Sciences Appliquées de Rennes, SC - Dépt. Signal et Communications, TELECOM Bretagne, 2003.

[34] J. Le Masson, "Systèmes de transmission avec précodage linéaire et traitement itératif," Ph.D. dissertation, ELEC - Dépt. Electronique, TELECOM Bretagne, 2005.

[35] M. Tuchler, R. Koetter, and A. C. Singer, "Turbo equalization: principles and new results," *IEEE Transactions on Communications*, vol. 50, no. 5, pp. 754–767, May 2002.

[36] C. Laot, R. Le Bidan, and D. Leroux, "Low-complexity MMSE turbo equalization: a possible solution for EDGE," *IEEE Transactions on Wireless Communications*, vol. 4, no. 3, pp. 965–974, May 2005.

[37] H. Omori, T. Asai, and T. Matsumoto, "A matched filter approximation for SC/MMSE iterative equalizers," *IEEE Communications Letters*, vol. 5, no. 7, pp. 310–312, July 2001.

[38] P. J. Bouvet, "Récepteurs itératifs pour systèmes multi-antennes," Ph.D. dissertation, INSA Rennes - Institut National des Sciences Appliquées de Rennes, Laboratoire Broadband Wireless Access, France Telecom division R&D, 2005.

[39] O. Muller, "Architectures multiprocesseurs monopuces génériques pour turbo-communications haut-débit," Ph.D. dissertation, TELECOM Bretagne, Department of Electronics, Brest, France, 2008.

[40] G. Masera, G. Piccinini, M. Roch, and M. Zamboni, "Vlsi architectures for turbo codes," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,*, vol. 7, no. 3, pp. 369–379, Sept. 1999.

[41] E. Boutillon, W. Gross, and P. Gulak, "Vlsi architectures for the map algorithm," *IEEE Transactions on Communications,*, vol. 51, no. 2, pp. 175–185, Feb. 2003.

[42] Y. Zhang and K. Parhi, "Parallel turbo decoding," in *Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. ISCAS '04.*, vol. 2, 23-26 May 2004, pp. II–509–12Vol.2.

[43] H. Moussa, "Architecture de Réseaux sur Puce Pour Décodeur Canal Multiprocesseurs," Ph.D. dissertation, ELEC - Dépt. Electronique, TELECOM Bretagne, 2009.

[44] J. Zhang and M. Fossorier, "Shuffled iterative decoding," *IEEE Transactions on Communications*, vol. 53, no. 2, pp. 209–213, Feb. 2005.

[45] P. Ienne and R. Leupers, *Customizable Embedded Processors–Design Technologies and Applications*, ser. Systems on Silicon Series. San Mateo, Calif.: Morgan Kaufmann, 2006.

[46] W. D. NEWCOM++ (NoE FP7), "Report on the state for the art on hardware architectures for flexible radio and intensive signal processing," *http://www.newcom-project.eu:8080/Plone/public-deliverables/research/DR.C.1_final.pdf*.

[47] "Coware processor designer homepage," *http://www.coware.com/products/processordesigner.php*.

[48] "Target ip designer homepage," *http://www.retarget.com/resources.php*.

[49] "Tensilica xtensa 7 homepage," *http://www.tensilica.com/products/x7_processor_generator.htm*.

[50] "Arc configurable cores homepage," *http://www.arc.com/configurablecores/*.

[51] "Stretch software-configurable processors homepage," *http://www.stretchinc.com/technology/*.

[52] B. Mei, A. Lambrechts, J.-Y. Mignolet, D. Verkest, and R. Lauwereins, "Architecture exploration for a reconfigurable architecture template," *IEEE Design Test of Computers*, vol. 22, no. 2, pp. 90 – 101, 2005.

[53] A. Hoffmann, O. Schliebusch, A. Nohl, G. Braun, O. Wahlen, and H. Meyr, "A methodology for the design of application specific instruction set processors (ASIP) using the machine description language lisa," in *IEEE/ACM International Conference on Computer Aided Design 2001. ICCAD 2001.*, 2001, pp. 625–630.

[54] M. Martina, G. Masera, H. Moussa, and A. Baghdadi, "On chip interconnects for multiprocessor turbo decoding architectures," *Microprocessors and Microsystems*, pp. –, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/B6V0X-50X2NGR-1/2/1a18b1777c8fb83f10b914b819d92ba9

[55] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Design, Automation and Test in Europe Conference and Exhibition*, 2000, pp. 250–256.

[56] W. J. Dally and B. Towels, "Route packets, not wires: On-chip interconnection networks," in *Design Automation Conference*, 2001, pp. 684–689.

[57] L. Benini and G. D. Micheli, "Networks on chips: a new soc paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, Jan 2002.

[58] S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *IEEE Computer Society Annual Symposium on VLSI*, 2002, pp. 105–112.

[59] L. Benini, "Application specific NoC design," in *Design, Automation and Test in Europe Conference and Exhibition*, 2006, pp. 1330–1335.

[60] F. Vacca, H. Moussa, A. Baghdadi, and G. Masera, "Flexible architectures for LDPC decoders based on network on chip paradigm," in *Euromicro Conference on Digital System Design*, 2009, pp. 582–589.

[61] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2003.

[62] A. Jantsch and H. Tenhunen, Eds., *Networks on chip*. Hingham, MA, USA: Kluwer Academic Publishers, 2003.

[63] A. Giulietti, L. van der Perre, and A. Strum, "Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements," *Electronics Letters*, vol. 38, no. 5, pp. 232 –234, 28 2002.

[64] C. Berrou, Y. Saouter, C. Douillard, S. Kerouedan, and M. Jezequel, "Designing good permutations for turbo codes: towards a single model," in *Communications, 2004 IEEE International Conference on*, vol. 1, 20-24 2004, pp. 341 – 345.

[65] A. Nimbalker, T. Blankenship, B. Classon, T. Fuja, and D. Costello, "Contention-free interleavers for high-throughput turbo decoding," *IEEE Transactions on Communications,*, vol. 56, no. 8, pp. 1258 –1267, august 2008.

[66] M. Martina and G. Masera, "Turbo noc: A framework for the design of network-on-chip-based turbo decoder architectures," *IEEE Transactions on Circuits and Systems I*, vol. 57, no. 10, pp. 2776 – 2789, 2010.

[67] M. Scarpellino, A. Singh, E. Boutillon, and G. Masera, "Reconfigurable architecture for ldpc and turbo decoding: A noc case study," in *IEEE 10th International Symposium on Spread Spectrum Techniques and Applications, 2008. ISSSTA '08.*, 25-28 2008, pp. 671 –676.

[68] H. Moussa, O. Muller, A. Baghdadi, and M. Jezequel, "Butterfly and benes-based on-chip communication networks for multiprocessor turbo decoding," in *Design, Automation Test in Europe Conference Exhibition, 2007. DATE '07*, 16-20 2007, pp. 1 –6.

[69] H. Moussa, A. Baghdadi, and M. Jezequel, "Binary de bruijn interconnection network for a flexible ldpc/turbo decoder," in *IEEE International Symposium on Circuits and Systems, 2008. ISCAS 2008.*, 18-21 2008, pp. 97 –100.

[70] O. Muller, A. Baghdadi, and M. Jezequel, "Exploring parallel processing levels for convolutional turbo decoding," *Information and Communication Technologies, ICTTA '06.*, vol. 2, pp. 2353–2358, 0-0 2006.

[71] H. Moussa, O. Muller, A. Jafri, O. Al-Aseel, G. Le Mestre, A. Baghdadi, and M. Jezequel, "Prototyping of multi-ASIP NoC-based architecture for exible turbo decoder," in *in Proc. of the conference on Design, Automation and Test in Europe, DATE'09*, April 2009.

[72] A. Hekstra, "An alternative to metric rescaling in viterbi decoders," *IEEE Transactions on Communications*, vol. 37, no. 11, pp. 1220–1222, 1989.

[73] A. Burg, M. Borgmann, M. Wenk, M. Zellweger, W. Fichtner, and H. Bolcskei, "Vlsi implementation of mimo detection using the sphere decoding algorithm," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 7, pp. 1566 – 1577, 2005.

[74] M. Karkooti, J. R. Cavallaro, and C. Dick, "FPGA Implementation of Matrix Inversion Using QRD-RLS Algorithm," in *Conference Record of the Thirty-Ninth Asilomar Conference on Signals, Systems and Computers*, October-November 2005, pp. 1625–1629.

[75] M. Myllyla, J. Hintikka, J. Cavallaro, and M. Juntti, "Complexity analysis of MMSE detector architecture for MIMO OFDM systems," *"in Proc. Asilomar Conf. on Signal, Systems and Computers."*, 2005.

[76] F. Edman and V. Öwall, "A scalable pipelined complex valued matrix inversion architecture," in *ISCAS'05, IEEE International Symposium on Circuits And Systems*, 2005, pp. 4489–4492.

[77] J. Eilert, D. Wu, and D. Liu, "Efficient complex matrix inversion for MIMO Software Defined Radio," *in Proc. IEEE International Symposium on Circuits and Systems.*, 2007.

[78] L. Boher, R. Rabineau, and M. Hélard, "Architecture and implmentation of an iterative receiver for MIMO systems," *"5th International Symposium on Turbo Codes and Realted Topics"*, Aug 2008.

[79] D. Karakolah, "Conception et prototypaged'un récepteur itératif pour des systèmes de transmision MIMO avec précodage linéaire," Ph.D. dissertation, ELEC - Dépt. Electronique (Institut Télécom-Télécom Bretagne), UBS - Université de Bretagne Sud (.), 2009.

[80] H. Kim, W. Zhu, J. Bhatia, K. Mohammad, A. Shah, and B. Danesrad, "A Practical Hardware Friendly MMSE Detector for MIMO-ODFM-Based Systems," *EURASIP Journal on Advances in Signal Processing*, 2008.

[81] K. A. Cavalec, G. Sicot, and D. Leroux, "Reduced complexity near-optimal iterative receiver for Wimax full-rate space time code," in *5th international symposium on Turbo Codes and related topics*, 2008.

[82] P.-J. BOUVET, "Récepteurs itératifs pour systèmes multi-antennes," Ph.D. dissertation, INSA de Rennes France, 2005.

[83] G. H. Golub and C. F. Van Loan, *Matrix Computations*. JohnsHopkinsPress, 1989.

[84] M. Myllyla, J. M. Hintikka, J. R. Cavallaro, M. Juntti, M. Limingoja, and A. Byman, "Complexity Analysis of MMSE Detector Architectures for MIMO OFDM Systems," in *Asilomar, Conference on Signals, Systems and Computers*, 2005, pp. 75–81.

[85] R. Döhler, "Squared Givens rotation," *IMA Journal of Numerical Analysis*, vol. 11, no. 1, pp. 1–5, 1991.

[86] J. E. Volder, "The Cordic Trigonometric Computing Technique," *IEEE Transactions on Electronic Computers*, vol. 8, no. 3, pp. 330–334, 1959.

[87] F. Edman, "Digital Hardware Aspects of Multiantenna Algorithms," Ph.D. dissertation, Lund University, Department of Electroscience, Lund, Sweden, 2006.

[88] D. Karakolah, C. Jégo, C. Langlais, and M. Jézéquel, "Design of an iterative receiver for linearly precoded mimo systems," *ISCAS09 : IEEE International Symposium on Circuits And Systems, Taiwan*, May 2009.

[89] F. Tosato and P. Bisaglia, "Simplified soft-output demapper for binary interleaved COFDM with application to HIPERLAN/2," *IEEE International Conference on Communications, 2002.*, pp. 664–668 vol.2, 2002.

[90] Altera, "Constellation mapper and demapper for wimax," *http://www.altera.com/literature/an/an439.pdf*, May 2007.

[91] L. Meng, C. Abdel Nour, C. Jégo, and C. Douillard, "Design of rotated QAM mapper/demapper for the DVB-T2 standard," *SiPS 2009 : IEEE workshop on Signal Processing Systems*, Oct. 2009.

List of publications

Book Chapter

[1] Abstract of book chapter "ASIP Design and Prototyping for Wireless Communication Applications", In "Rapid Prototyping" book, InTech Open Access Publisher, accepted, 2011.

Journals

[2] A.R. Jafri, A. Baghdadi, and M. Jézéquel, "ASIP-based Universal Demapper for Multiwireless Standards", *The inauguration issue of IEEE Embedded Systems Letters*, vol. 1, no. 1, pp. 9-13, May 2009.

[3] A.R. Jafri, A. Baghdadi, and M. Jézéquel, "Parallel MIMO Turbo Equalization", *IEEE Communication Letters*, accepted for publication, to appear, 2010.

[4] A.R. Jafri, A. Baghdadi, and M. Jézéquel, "EquASIP: A Universal MMSE-IC Linear Equalizer for Multi Wireless Standards", *IEEE Transactions on Circuits and Systems I*, submitted, under revision, 2010.

[5] A.R. Jafri, A. Baghdadi, and M. Jézéquel, "From Parallelism to Heterogeneous multi-ASIP and NoC based Implementation of Turbo Demodulation", under preparation.

[6] A.R. Jafri, A. Baghdadi, and M. Jézéquel, "A Heterogeneous multi-ASIP and NoC based Unified Turbo Receiver for Multi Wireless Standards", under preparation.

Conferences

[7] A.R. Jafri, D. Karakolah, A. Baghdadi, and M. Jézéquel, "ASIP-based Flexible MMSE-IC Linear Equalizer for MIMO Turbo-Equalization Applications", *In Proc. IEEE/ACM Design, Automation and Test in Europe Conference & Exhibition, DATE'09*, Nice, France, 21-23 April 2009.

[8] H.Moussa, O. Muller, A.R. Jafri,A. Baghdadi, J. Le Mestre, and M. Jézéquel, "FPGA Prototypes For Turbo Communication Applications", *Poster and Demonstration at the University Booth of the IEEE/ACM Design, Automation and Test in Europe Conference & Exhibition, DATE'09*, Nice, France, 21-23 April 2009.

[9] A.R. Jafri, A. Baghdadi, and M. Jézéquel, "Rapid Prototyping of ASIP-based flexible MMSE-IC Linear Equalizer", *In Proc. IEEE/IFIP 20th International Symposium on Rapid System Prototyping, RSP'09*, Paris, France, 23-26 June 2009.

[10] A.R. Jafri, A. Baghdadi, and M. Jézéquel, “Rapid Design and Prototyping of Universal Soft Demapper”, *In Proc. IEEE International Symposium on Circuits and Systems, ISCAS’10*, Paris, France, 30 May - 2 June 2010.

[11] A.R. Jafri, A. Baghdadi, and M. Jézéquel, “DemASIP : Universal Demapper for Multiwireless Standards”, *In GDR SoC-SiP: Groupe de recherche System on Chip - System in Package, Colloque National*, Paris, France, 9-11 June 2010.

[12] A.R. Jafri, A. Baghdadi, and M. Jézéquel, “Exploring Parallel Processing Levels in Turbo Demodulation”, *In Proc. IEEE 6th International Symposium on Turbo Codes and Iterative Information Processing*, Brest, France, 6-10 Sept 2010.

[13] A.R. Jafri, A. Baghdadi, and M. Jézéquel, “FPGA Prototypes of Heterogeneous multi-ASIP and NoC based Unified Turbo Receiver for Multi Wireless Standards”, accepted, *Poster and Demonstration at the University Booth of the IEEE/ACM Design, Automation and Test in Europe Conference & Exhibition, DATE’11*, Grenoble, France, 14-18 March 2011.